

OPERATING SYSTEMS

: 0012 chapter 2 all 0

Android, IOS, Symbios, ~~SUM~~, WINDOWS, Linux

Uses:-

Without OS there is no use of system.

Use → Proper management

OS is a system software by the use of system slw we can bring the h/w in an environment in which user can operate accordingly.

h/w → memory, processor, peripheral devices.

Application slw → Apps

System slw

OS is a system slw.

By the use of system slw we can bring the h/w into a working environment so that the user can work in a systematic & reliable manner.

The OS controls the way slw uses h/w.

The purpose of this control is to make the computer operate in the way intended by the user, and in a systematic reliable & efficient manner.

Other system s/w:

→ Compiler

→ Assembler

→ Debugger

What is an OS

A program that acts as an intermediary b/w a user of a computer & computer h/w.

OS goals →

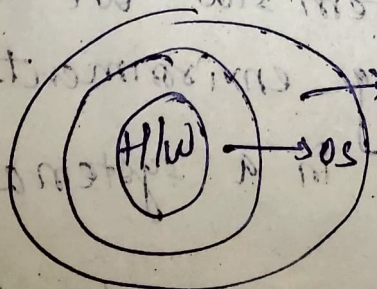
→ Execute user program & make solving user problem

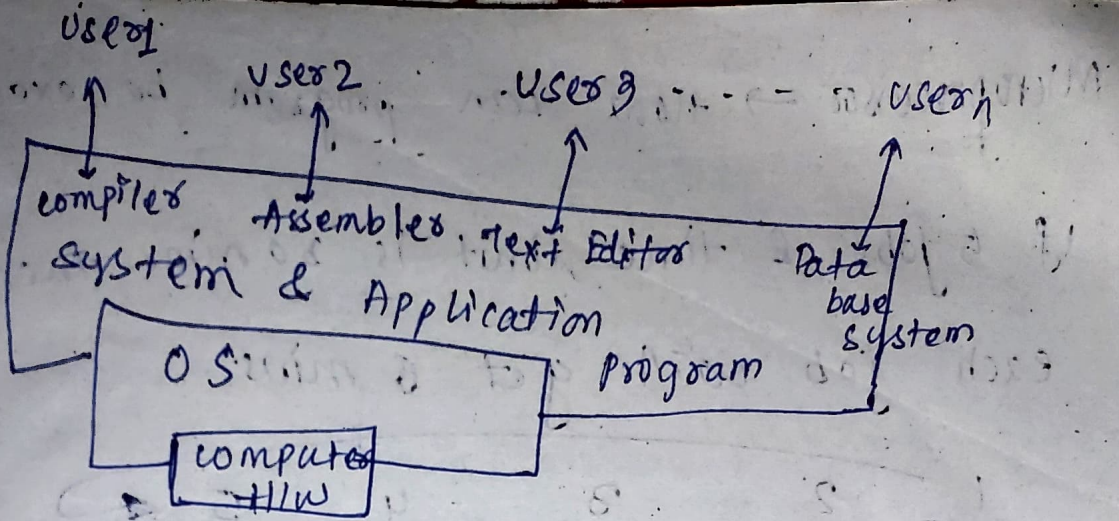
→

→

Abstract view of system

Four components of computer system





OS is ~~also~~ stored in memory.

Instructions → functions of OS

Functions of OS:-

① Command Interpreter :-

OS works as command interpreter

Every command that is typed by the user is interpreted by the OS. After the interpreting, action is taken accordingly to the command.

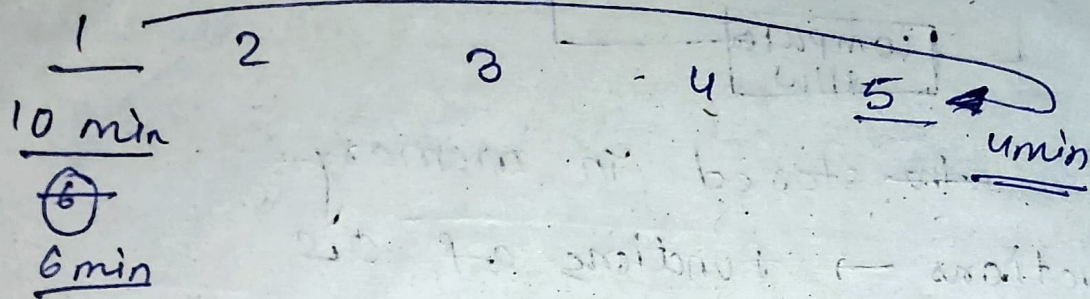
For eg: when the user types date, the system interprets it & prints the date & time in terminal.

② Time sharing :-

This is way to allow more than one user to use the system at a time. While many users are using the system/computer simultaneously, the OS ensures that their jobs don't get mixed, with each other.

Microprocessors $\rightarrow$ How can program be executed.

If 5 jobs & time limit is 30 mins. then each job will get 6 mins.



The 6 min will be utilized & remaining 4 min will go to tail of the job.

(3) Security: The OS protects one user from another user. No user is allowed to see the information manipulated by another user.

(4) Programme Development Tools:

The tools are generally programs written by the system administrator & experts. They help the user to develop their own program in a better & neater & easier fashion. The collection of such tools provides

an environment in which an ordinary user feels more comfortable.

⑤ Revolve tools

Y2K

Date → DD MM YYYY

£ 5000 →

92/5/4

£ 10000 →

Spells

97/5/4

⑤

05/04/98

05/04/03

⑤

④④

④④④④

Revolve tools

converted

74 into 777

2 digit to 4 digit

⑥ Communication Facilities :-

OS also provides the facilities to communicate with each other. These days interlinked technically, we say ~~all~~ n/ws are build up for communication. Using these facilities OS communicate with each other & let their user also communicate.

i.e. windows and a diff OS can communicate with another diff. OS like UBUNTU

⑥ I/O & O/P:

Through mouse, keyboard & many other devices the user communicates with the system such as disk, tape, through which the info. is supplied to (i/p) and obtained from (o/p) the computer. The OS ensures smooth functioning of all these ~~per~~ peripherals.

⑦ Info. management & accounting:

Every user of computer system wants to manipulate some kind of info. The OS provides ways to store & manipulate this info. The OS facilitates the collection of various statistics that helps in improving info.

Overlay

Spooling

Buffering

Time sharing

Privilege instruction

Real time systems

~~Short notes~~

Short notes

Components of OS:-

→ 4 components:-

- ① Memory management → How to store in memory.
- ② Device → Peripheral Devices before execution
- ③ Process → Prioritise any job.
- ④ Information → choose & executes job

08/01/24

Operating Systems

Completed → Defn of OS

→ func of OS

→ Short notes

multiprogramming

1. Privileged instruction
2. Spooling
3. Buffering
4. Multiprogramming
5. Overlays

Components of OS

- memory management
- Device
- Process
- Information

Functions of memory management

- ① Keep track of resources/memory.
(memory is resource)
- ② In a multiprogramming environment it decides who have a chance to get resource.

Process

task

Program

Jobs

Same thing.

5 jobs.

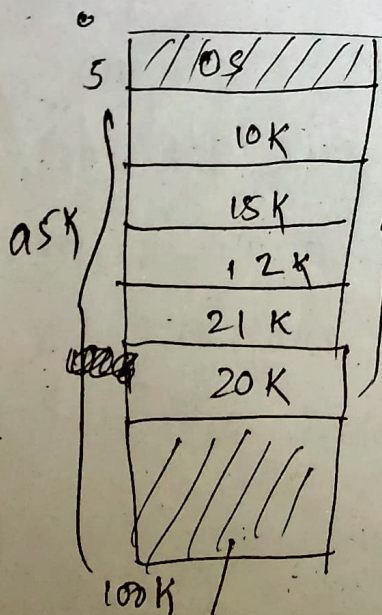
who will get the chance to come first and then second and accordingly i.e.

who will get the memory 1st

Program prior to execution it stores in

~~Secondary memory~~ DSD Primary memory and after that in secondary memory.

The actual storage is not for user, some of the storage is occupied by OS, rest is with user area.



5 jobs

1 2 3 4 5

storage $\rightarrow 1+2+3+4+5$

jobs
795K

who gets the storage 1st is decided by memory management

Suppose storage by 5 jobs are 2-

Jobs: 1, 2, 3, 4, 5

$\Rightarrow 72K, 30K, 23K, 21K, 10K \Rightarrow ?$

$\rightarrow$ If 15K job arises it can be allocated or 1640 17K

some memory
10K
(17K)

How to allocate jobs such that it should be in a efficient manner.

If 1st job is given then it will ~~get~~ take more time & other will ~~sto~~ suffer starvation.

So, Job 1 & 3 ~~is~~ can be allocated 1st which total makes up 95k. it can be next deallocated & next is allocated.

③ Allocate the resource.

④ Deallocate the resource when the job will be over. (Reclaim)

1st job is executed and after 2nd job is then allocated which before execution it detects the 1/p o/p device is required.

Device management → Resource is device.

Information management -

① VP data is valid or not

Among 3 jobs whose info. can be visible 1st is first ~~also~~ executed

Process management :-

which job will be executed 1st

Either in shortest job 1st

or in 1st come 1st serve

or Priority basis

Problem persons

~~OS~~

mendick & Dollovan

Book

Mathem
actual

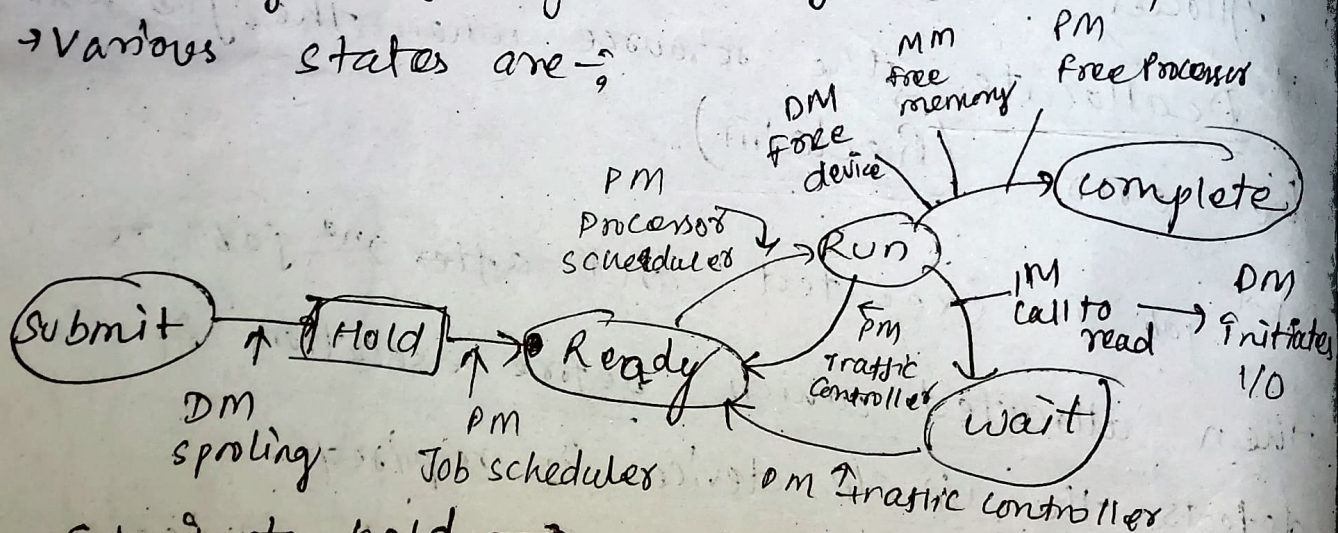
Theoretical → Peterson, Galvin

Imp unmark

Life cycle of process :- (or process state)

→ Submitting a job goes through various process

→ Various states are :-



Submit to hold → submit → everything is ready
Hold

Ready → checked parameters

Spooling → simultaneous peripheral operation
online.

↓
side by side

Job scheduler $\rightarrow$ ~~Arranges~~ D.K patcher (P.C.F.S)

Process scheduler $\rightarrow$ Allocated to processor.

where processor is free or not

5 Jobs
(Uoms)

generated by OS
 $\downarrow$
8ms each
if equally divided

1 $\rightarrow$ 20K $\rightarrow$ Job 1 is executed

2 $\rightarrow$ 12K

3 $\rightarrow$ 13K

4 $\rightarrow$ 14K

5 $\rightarrow$ 25K

PM $\rightarrow$ Process manager

MM $\rightarrow$ Memory

D.M $\rightarrow$ Device

At time of execution ~~if~~ if no work is needed
then it will go from RUN state to
WAIT state.

Multiprogramming $\rightarrow$ if more than 1 program executes
concurrently / simultaneously

on what basis?

if one processor is present $\leftarrow$

it can be done $\rightarrow$ By creating

multi programming environment.

Either by more than 1 processor.

if single processor than by time sharing

the program execution is one by one but it

executes so fast that it goes concurrently.
When job is not executed then it goes
from RUN state to WAIT state. And go
to the tail of ~~q~~ & goes to ready state.
If all jobs are completed and one is
still present in READY state and will
execute when other jobs are arriving or
assigned. → single processor.

Submit → similar types of program are placed
together. & deployed at a time.

Spooling → By this we bring all the program
in hold state.

HOLD → It is a huge storage area.

DASD → Direct Access storage Device.

MVS → OS for main frame. (secondary storage)

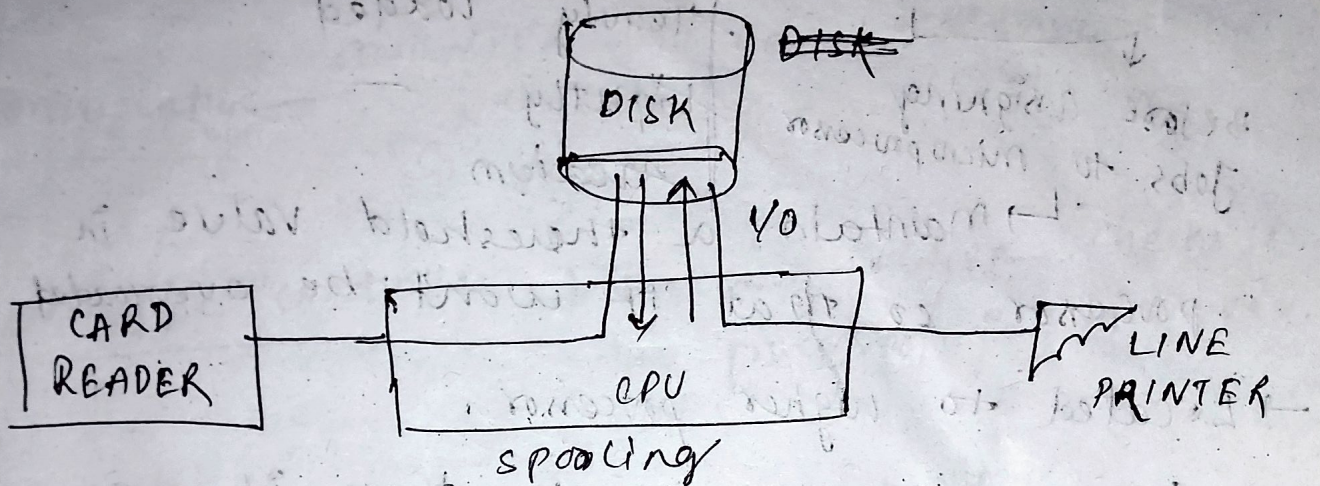
JCL → Job control language.

White Box Testing → Program is imp. ^{visible to} user.

Black → No need to go for
internal program & o/p
is main

Spooling → it is acronym for simultaneous peripheral operation online.

→ computation of one program & o/p may be result of another program.



card reader → Reads the submitted program.

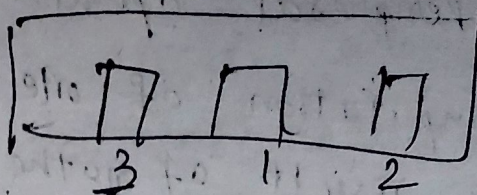
Spooling is done with magnetic tape. It is possible to read the entire content of tape into a disk.

Spooling → online operation multiprogram simultaneously.

Submit → Hold through spooling.

multiprocessor ← multiprogramming

6 Jobs



Processors

before assigning
Jobs to microprocessors.

Heavily loaded
lightly
medium

→ maintain a threshold value in
processor so that it won't be overloaded

→ Located to higher processor.

→ If one gets run out then its ~~jobs~~
Jobs are shared among other processors.

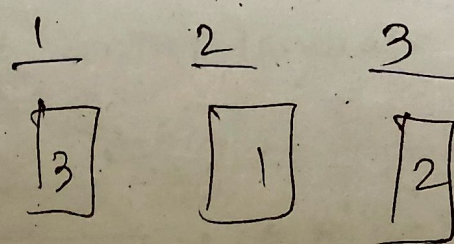
6 Jobs → 1 job → 1st job run out

shared among
at another
processor

2 jobs

Half

→ Returns to READY
state.



1 2
2 4

→ more efficient
Threshold Value

Multiprogramming → 1 processor → OS will generate time stamp
 ↓ (time sharing)

one after another job will be executed
 So fast.
 on currently ← some time

Traffic controller → processor is free checks if the processor is free or not.

~~processor~~
RUN → WAIT → for device checking
 traffic controller checks
 READY ← WAIT ← true/false

Privileged Instruction or Information →

JCB → Job control Bus

Ex: Halt Instruction
 ↓
 User mode
 ↓
 Trap will occur
 ↓
 immediately it will go to OS monitor mode.

OS mode → slow OS

sense register → slow

↓
Bus register

Few instructions are there that can be executed only on monitor or in OS

← user area

If we try to execute that in user mode a trap will occur.

Spooling → ONLine

buffering → off Line

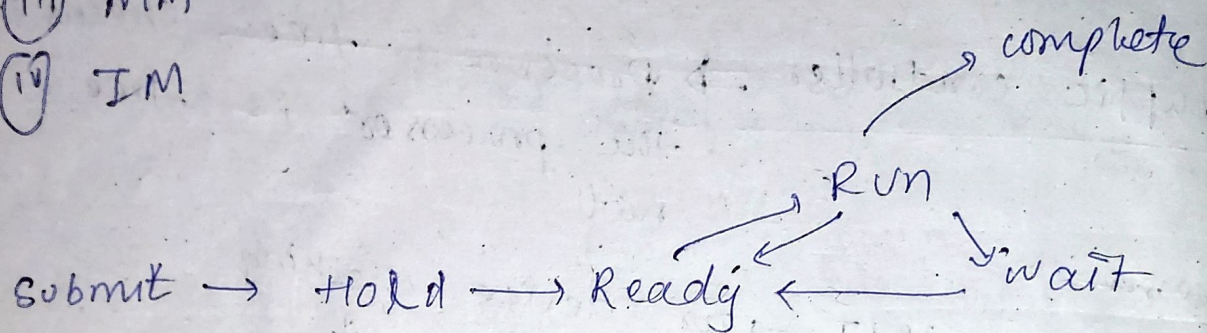
Component of OS :-

(I) PM

(II) DM

(III) MM

(IV) IM



{ Short term scheduler
long
medium

→ in PM

// Submit → submitting batch of jobs

Job controller controls the job.

All jobs are preceded by job controller cards.

~~Hold state~~ Hold state by spooling
from submit state.

In Hold state ~~we~~ consist of DASD we

check the peripheral devices required &

memory management also IM & PM + that it
is free or not then proceeds to Ready state.

Time taken for all the management i.e.

submit to READY state → long term

and done by job scheduler is known
as long term scheduler.

Job scheduler → Decides the sequence of
management i.e. PM, IM,
MM

CPU → short term scheduler
scheduler.

~~Medium~~ Medium term scheduler → ~~R~~
Ready to wait → when it ~~crashes~~ down
windown or run out.

Process management :-

In which sequence job will be allocated to execute.

Scheduling → OS schedule in which sequence jobs will be executed.

~~Non-preemptive~~

~~Non-primitive~~

Non-preemptive



once the scheduling decision has been made by any means the decision is not going to change.

↳ FCFS → Job arriving

↳ SJF → will be executed 1st.

3. Priority

(waiting & response time are same)

~~Preemptive~~

~~Primitive~~

Preemptive



means forcefulness



Decision can be changed even after decision has been made.

↳ RR → Round roving

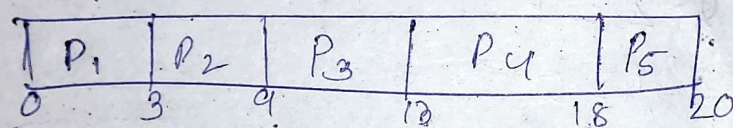


Time sharing

Q. → It is the time taken by the CPU to execute 1 job

Process	(min.) CPU burst	Arrival time	Job
P ₁	3	0	FCFS
P ₂	6	2	
P ₃	4	4	
P ₄	5	6	
P ₅	2	8	

Gantt chart



Turn around time = finish time - Arrival time

$$P_1 = 3 - 0 = 3$$

$$P_2 = 9 - 2 = 7$$

$$P_3 = 13 - 4 = 9$$

$$P_4 = 18 - 6 = 12$$

$$P_5 = 20 - 8 = 12$$

$$\frac{12}{5} = 2.4$$

Avg. turn around time (TAT)

Relative delay = $\frac{\text{Turn around time}}{\text{CPU burst time}}$

$$P_1 = 3/3 = 1$$

$$P_2 = 7/6 = 1.17$$

$$P_3 = 9/4 = 2.25$$

$$P_4 = 12/5 = 2.40$$

$$P_5 = 12/2 = 6$$

$$\text{Avg. Relative} = \frac{2}{5} = 2.56$$

Waiting time = Start time - Arrival time.

$$P_1 = 0 - 0 = 0$$

$$P_2 = 3 - 2 = 1$$

$$P_3 = 9 - 4 = 5$$

$$P_4 = 13 - 6 = 7$$

$$P_5 = 18 - 8 = 10$$

Avg. waiting time
 $= 23/5 = 4.6 \text{ MSec}$

Response Time = First Response - Arrival Time

$$P_1 = 0 - 0 = 0$$

$$P_2 = 3 - 2 = 1$$

$$P_3 = 9 - 4 = 5$$

$$P_4 = 13 - 6 = 7$$

$$P_5 = 18 - 8 = 10$$

First Response $\rightarrow$ When particular job starts.

Q1.

Process	Burst time
P_1	5
P_2	24
P_3	16
P_4	10
P_5	3

Q2.

Process	Burst time	AT
P_1	3	0
P_2	6	2
P_3	4	4
P_4	5	6
P_5	2	8

AT $\rightarrow$ not given $\rightarrow$ then starts from 0.

$\hookrightarrow$ Arrival time.

Process State or Life Cycle.

①

- A user submits job to the system by placing his deck into a card reader (SUBMIT State) the job consist of several decks of program preceded by job control cards. (No. of job cards)
- The job control cards pass information to the OS as to what resource the job will need.
- The spooling routine ~~reads~~ ^{reads} the job and place it into a disk. (HOLD state) or DASD
- The job scheduler determines the device requirements from the users jobs ~~part~~ cards.
- Once the job scheduler decideds that the jobs has to be assigned the required devices or resources, the traffic controller is called to create the associated process info. and memory management is called to allocate the necessary main memory.
- The job is then loaded into memory & the process is ready to run (READY State)

②

→ When a processor becomes free, the process scheduler scans the list of process ready processes and choose a process and assigns a processor to it (RUNNING state)

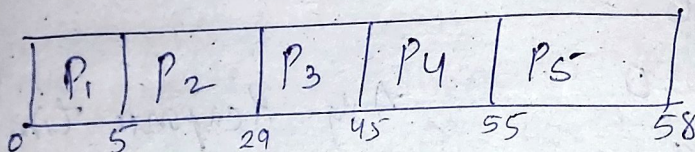
→ If the running process request access to read a file, IM will call DM to initiate the reading of the file. → DM would initiate the I/O operation and then call PM to indicate that the process requesting the file is awaiting the completion of I/O operation (WAIT state).

→ When the I/O is completed the traffic controller places the process back into the READY state. ~~if the~~

→ If the process complete its computation then it will be placed into complete state and the allocated resources will be free.

<u>Process</u>	<u>Burst Time</u>	<u>Arrival Time</u>
P ₁	5	0
P ₂	24	5
P ₃	16	29
P ₄	10	45
P ₅	3	55

Gantt chart



Turn around time = Finish time - Arrival time

$$P_1 = 5 - 0 = 5$$

$$P_2 = 29 - 5 = 24$$

$$P_3 = 45 - 29 = 16$$

$$P_4 = 55 - 45 = 10$$

$$P_5 = 58 - 55 = 3$$

$$\frac{58}{5} = 11.6 \text{ ms} = \text{Avg Turn Around Time}$$

Relative delay = $\frac{\text{Turn around time}}{\text{CPU burst time}}$

$$P_1 = 5/5 = 1$$

$$P_2 = 24/24 = 1$$

$$P_3 = 16/16 = 1$$

$$P_4 = 10/10 = 1$$

$$P_5 = 3/3 = 1$$

Avg Relative delay = 1.

Waiting Time = Start time - Arrival Time

$$P_1 = 0 - 0 = 0$$

$$P_2 = 5 - 5 = 0$$

$$P_3 = 29 - 29 = 0$$

$$P_4 = 45 - 45 = 0$$

$$P_5 = 55 - 55 = 0$$

Avg. waiting time = 0

Response Time = First Response - Arrival Time

$$P_1 = 0 - 0 = 0$$

$$P_2 = 5 - 5 = 0$$

$$P_3 = 29 - 29 = 0$$

$$P_4 = 45 - 45 = 0$$

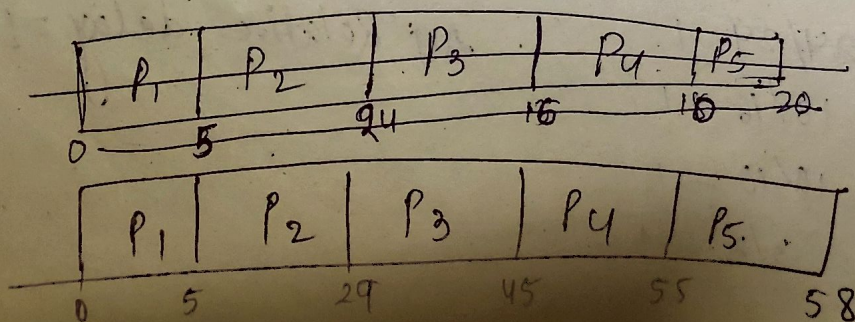
$$P_5 = 55 - 55 = 0$$

Avg. Response time = 0

Q2:

Process	CPU burst time	Arrival Time
P ₁	3	0
P ₂	24 24	0
P ₃	4 16	0
P ₄	5 10	0
P ₅	3	80

Grant Chart



Turn around = finish time - Arrival time

$$P_1 = 5 - 0 = 5$$

$$P_2 = 29 - 0 = 29$$

$$P_3 = 16 - 0 = 16$$

$$P_4 = 10 - 0 = 10$$

$$P_5 = 3 - 0 = 3$$

$$5 - 0 = 5$$

$$29 - 0 = 29$$

$$45$$

$$55$$

$$58$$

$$\text{Avg. TAT} = \frac{5 + 29 + 45 + 55 + 58}{5} = \frac{192}{5} = 38.4$$

Relative Delay = $\frac{\text{Turn around Time}}{\text{CPU burst time}}$

$$P_1 = 5/5 = 1$$

$$P_2 = 29/24 = 1.208$$

$$P_3 = 45/16 = 2.8125$$

$$P_4 = 55/10 = 5.5$$

$$P_5 = 58/3 = 19.3333$$

Waiting time = Start time - Arrival time

$$P_1 = 0 - 0 = 0$$

$$P_2 = 5 - 0 = 5$$

$$P_3 = 29 - 0 = 29$$

$$P_4 = 45 - 0 = 45$$

$$P_5 = 55 - 0 = 55$$

$$\text{Avg. Waiting time} = \frac{0 + 5 + 29 + 45 + 55}{5} = \frac{134}{5} = 26.8$$

Response Time = First Response - Arrival Time

$$P_1 = 0 - 0 = 0$$

$$P_2 = 5 - 0 = 5$$

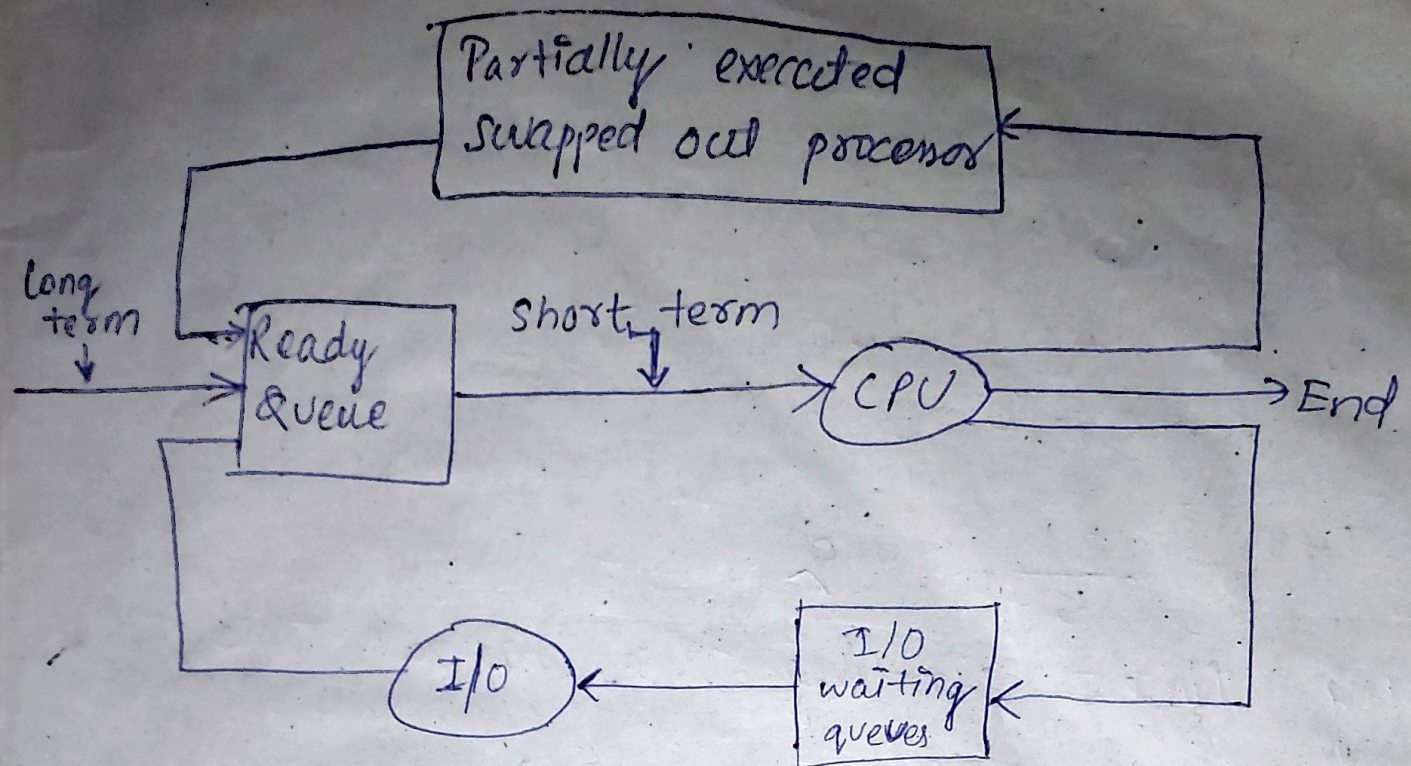
$$P_3 = 29 - 0 = 29$$

$$P_4 = 45 - 0 = 45$$

$$P_5 = 55 - 0 = 55$$

$$\text{Avg. Response Time} = \frac{0 + 5 + 29 + 45 + 55}{5} = \frac{134}{5}$$

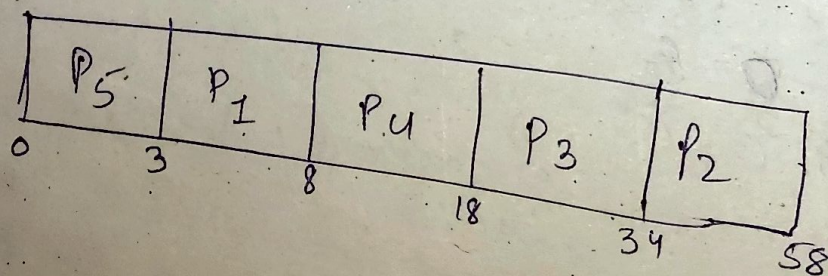
$$= 26.8$$



Q/ SJF $\rightarrow$ shortest CPU burst time

<u>Process</u>	<u>CPU Burst</u>
P ₁	5
P ₂	24
P ₃	16
P ₄	10
P ₅	3

Gantt Chart



Turn around time = Finish Time - Arrival Time

$$P_1 = 8 - 0 = 8$$

$$P_2 = 58 - 0 = 58$$

$$P_3 = 34 - 0 = 34$$

$$P_4 = 18 - 0 = 18$$

$$P_5 = 3 - 0 = 3$$

$$\text{Avg. TAT} = \frac{121}{5} = 24.2$$

Waiting time = Starting time - Arrival Time

$$P_1 = 3 - 0 = 3$$

$$P_2 = 34 - 0 = 34$$

$$P_3 = 18 - 0 = 18$$

$$P_4 = 8 - 0 = 8$$

$$P_5 = 0 - 0 = 0$$

$$\text{Avg. waiting time} = \frac{63}{5} = 12.6$$

Response Time = First Response - Arrival time

$$P_1 = 3 - 0 = 3$$

$$P_2 = 34 - 0 = 34$$

$$P_3 = 18 - 0 = 18$$

$$P_4 = 8 - 0 = 8$$

$$P_5 = 0 - 0 = 0$$

→ In priority scheduling, a priority is associated with each job & CPU is allocated to job with highest priority.

→ SJF → One of priority algorithm, where the priority P is inverse of CPU burst time (η).

(Time required for priority processing by)
$$CPU = \frac{1}{\eta} = P$$

Larger the CPU burst time lower the priority & vice versa.

→ Priority can be defined either internally or externally, internally define priorities uses measurable qty, i.e. time limits, memory requirement and ratio of the avg. I/O burst to avg CPU burst.

→ Externally priority is defined as the type and amount of funds being ~~for~~ paid for computers the department sponsoring the work and also political pressure.

Disadv.

Major disadvantages with priority scheduling algⁿ

→ Indefinite blocking (Starvation)

→ A PS algⁿ can leave some low priority process ~~which~~ waiting indefinitely for CPU.

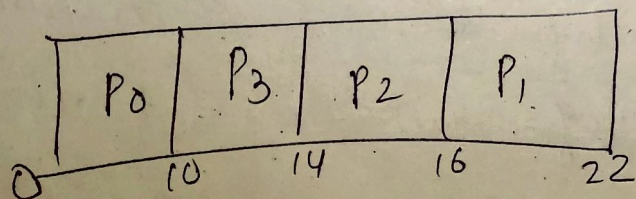
→ A solⁿ for indefinite blocking is aging.

→ It is a technique, which gradually increases the priority of job that waits in a system for long time. → aging technique

Non-Preemptive (Priority Boost) also

<u>Process</u>	<u>AT (ms)</u>	<u>Next Burst</u>	<u>Priorities</u>
P ₀	0	10	5
P ₁	1	6	4
P ₂	3	2	2
P ₃	5	4	0

Gantt chart



Finish Time

TAT

WT

10

10

0

22

21

15

16

13

11

14

9

5

avg. TAT

$$= \frac{53}{4}$$

$$= 13.25$$

avg. WT = $\frac{31}{4}$

$$= 7.75$$

Aging Technique →

scheduling →

Non-Preemptive

preemptive

FCFS
SJF
priority

Time sharing

complete

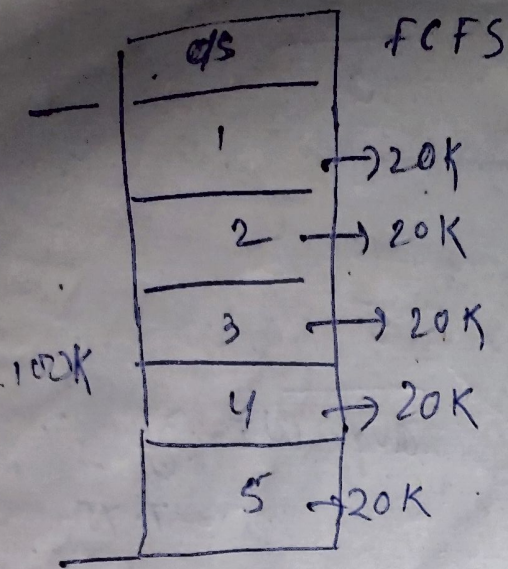
Submit → hold → Ready → wait

run

Run → Ready ⇒ scheduling

SJF → CPU burst time min. then greater priority.

Run → Ready ⇒ Jobs pulled out.



5 Jobs $\rightarrow$ equal $\rightarrow$ 20K space

Time space Trade off

$\rightarrow$ Time space complexity

Time complexity

Priority $\leftarrow$

Priority based on time taken.

Jobs executed in FCFS & SJF $\rightarrow$ The job in last priority of FCFS $\rightarrow$ may suffer starvation

Aging Technique $\rightarrow$ To overcome starvation, a job is executed sooner is called aging technique.

Time complexity $\rightarrow$ Internal priority

External Priority $\rightarrow$ Funds

PCB / JCB / TCB $\rightarrow$ Process control block
Job _____
Task _____

Political Pressure $\rightarrow$ Geographical Region

JCB $\rightarrow$ deals with external priority & internal priority.

Internal Priority is considered.
 when Int. priority can't be decided then ext. priority is considered.

Preemptive $\rightarrow$ Decision can be changed once scheduling is done.

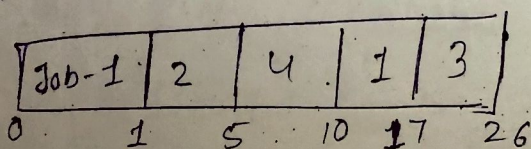
$\hookrightarrow$ If high priority job arrives then priority is given to that job.
 high priority job preempts the LP.

Non-preemptive $\rightarrow$ Decision can not be changed

Preemptive

Job	Arrival Time	Burst Time	Waiting time	Turn around Time	Finish time - Arrival time
1	0	8	$(8-0=7)$	$17-0=17$	
2	1	4	$(4-1=3)$	$5-1=4$	
3	2	9	$(15-2=13)$	$26-2=24$	
4	3	5	$(10-3=7)$	$10-3=7$	
			Avg. = $\frac{17+4+24+7}{4} = \frac{52}{4} = 13$		

Gantt Chart



$$\text{Avg. TAT} = \frac{(17-0) + (5-1) + (26-2) + (10-3)}{4} = \frac{52}{4} = 13$$

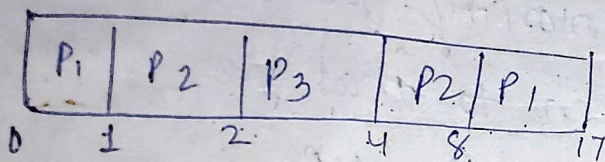
Q (SJF)

<u>Process</u>	<u>CPU burst time</u>	<u>Priority</u>	<u>Arrival Time</u>	<u>Turn around time</u> = $\frac{\text{Finish time} - \text{Arrival time}}{\text{Arrival time}}$
P ₁	10 (10-1=9)	3	0	17-0=17
P ₂	5 (5-1=4)	2	1	8-1=7
P ₃	2	1	2	4-2=2

$$\text{Avg. TAT} = \frac{17+7+2}{3}$$

$$= \frac{26}{3}$$

$$= 8.66$$



Relative delay

$$= \frac{\text{TAT}}{\text{CPU burst time}}$$

$$1.7$$

$$1.04$$

$$1$$

Avg. Relative

$$\text{Delay} = \frac{4.3}{3}$$

Waiter time

$$= \frac{\text{Start time} - \text{Arrival time}}{\text{Arrival time}}$$

$$0-0=0 \quad 8-1=7$$

$$1-1=0 \quad 4-2=2$$

$$2-2=0$$

$$\text{Avg. } \frac{9}{3} = 3$$

SCHEDULING

(3)

→ Long term scheduling:-

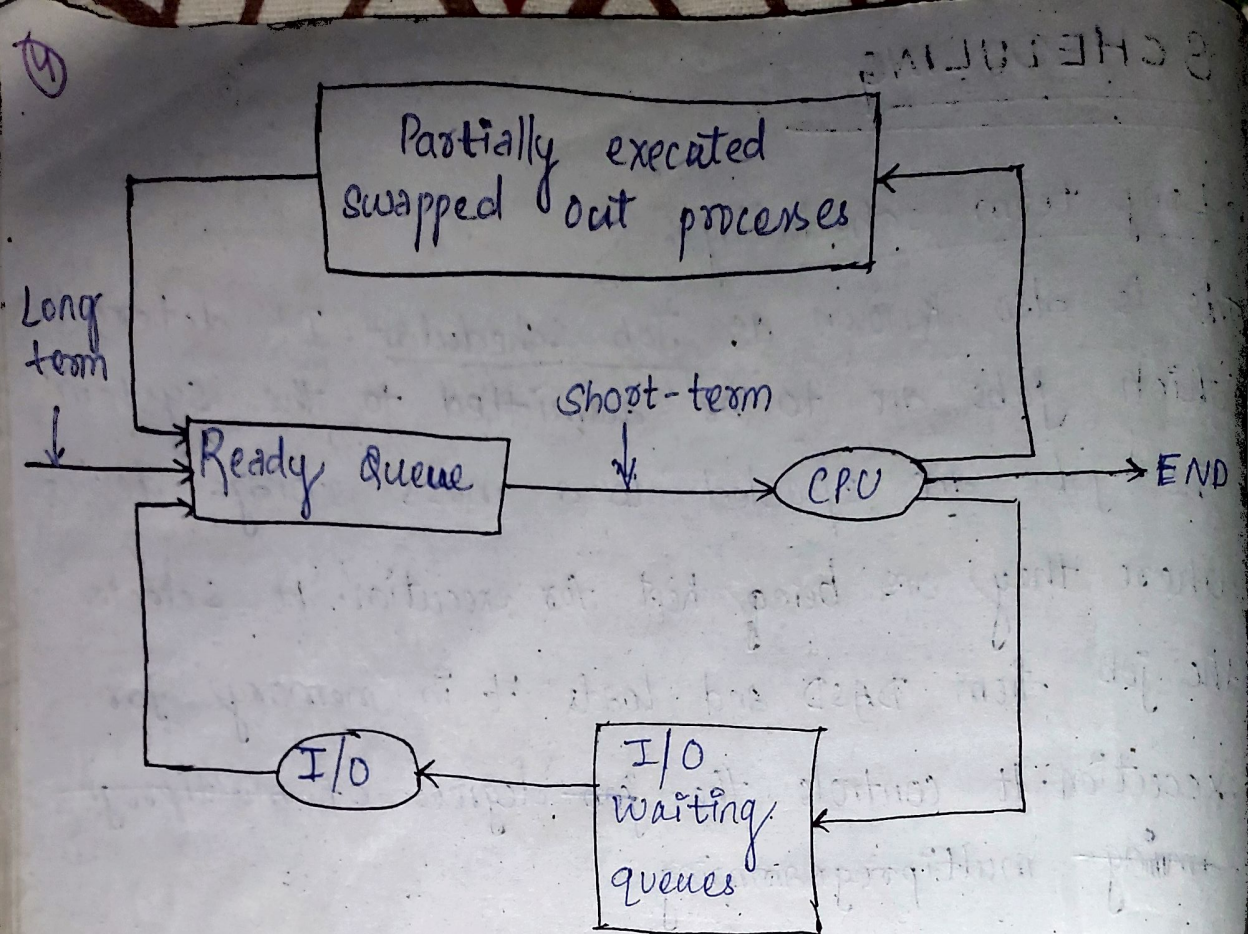
- This is also known as Job Scheduler. It determines which jobs are to be admitted to the system.
- The jobs are spooled to a mass storage device where they are being kept for execution. It selects the job from DASD and loads it in memory for execution. It controls the ~~job~~ degree of ~~multiprogramming~~ ~~among~~ multiprogramming.

→ Short term scheduling:-

- It is also known as CPU scheduling. It selects one job from the set of jobs from main memory which are ready to execute and allocate the CPU to one of them. This short-term scheduler must be very fast.

→ Medium term scheduling:-

- Medium term scheduler is the intermediate level of scheduling. It is used in time sharing operating systems. The main advantages of medium term scheduler is that it reduces the degree of multiprogramming, by swapping.



Non-Preemptive Scheduling:

- In non-preemptive scheduling, a scheduled job always completes before another scheduling decision is made. The jobs therefore finish in the order in which they are scheduled.
- The example of non-preemptive scheduling are
 - (i) First come first serve (FCFS)
 - (ii) Shortest job first (SJF)
 - (iii) Priority Algorithms.

(i) First Come First Serve (FCFS) :-

(5)

Algorithm

Here jobs (Process) which requests the CPU first is allocated the CPU first. So the implementation of FCFS is easily managed with FIFO queue.

The scheduler simply maintains a queue of pending jobs and schedules the first one it finds in the queue.

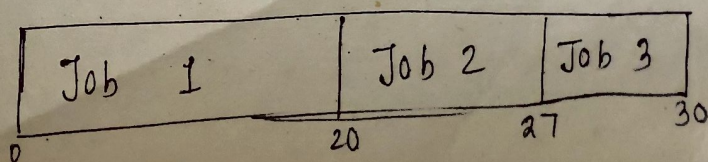
Performance

The performance of FCFS is often quite poor. The first job gets the best turnaround while the last job suffers the worst turn around.

Example :-

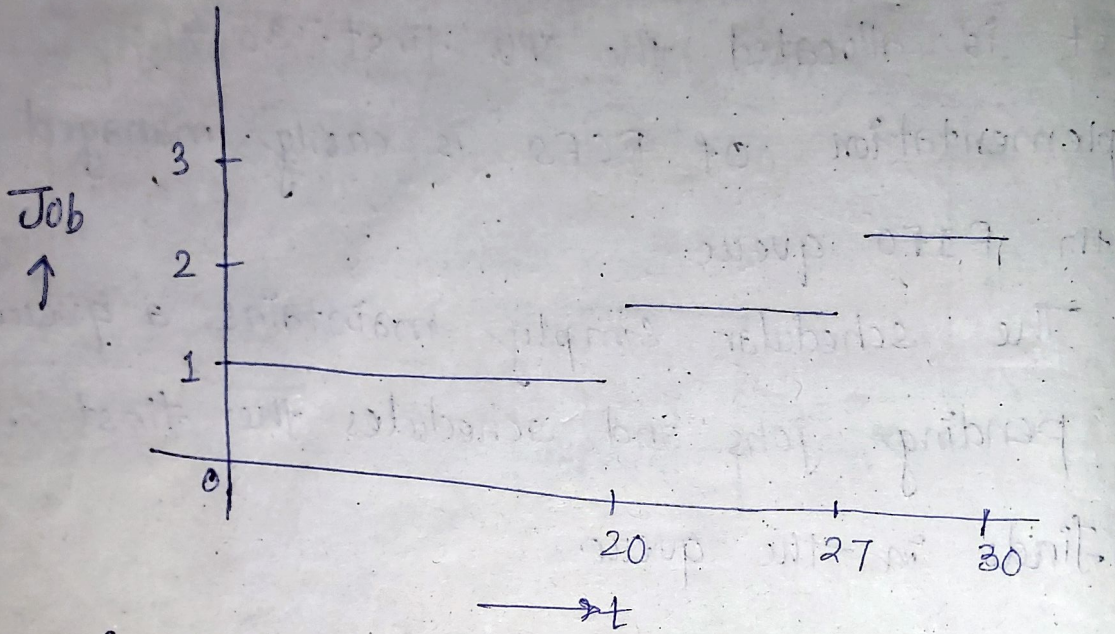
<u>Job</u>	<u>Burst Time</u>
1	20
2	7
3	3

If the job arrive in the order 1, 2, 3 then the Gantt chart.



The turnaround time for Job 1 is 20, Job 2 is 27 and Job 3 is 30.

So the average turnaround time is $\frac{20+27+30}{3} = 26$



Graphical Representation

Disadvantage: Convoy effect

(ii) Shortest - Job - First

Shortest job first associates with each job the length of its next CPU burst. When the CPU is available, it is assigned to that job with the smallest next CPU burst. If two jobs have the same next CPU burst then FCFS is used.

(iii) Priority

(7)

In priority scheduling a priority is associated with each job and the CPU is allocated to the job with the highest priority.

SJF is one type of priority algorithm where the priority (P) is the inverse of the CPU burst time (λ) [The time required for processing by CPU].

$$P = 1/\lambda$$

So the larger the CPU burst, the lower the priority and vice versa.

Priority can be defined either internally (or) externally. Internally defined priorities use measurable quantities i.e. the time limits, memory requirement and the ratio of average I/O burst to average CPU burst.

Externally priority is defined as the type and amount of funds being paid for computer, the department sponsoring the work, and also political pressures.

8)

Disadvantages:

- A major disadvantages with priority scheduling algorithms is indefinite blocking or starvation.

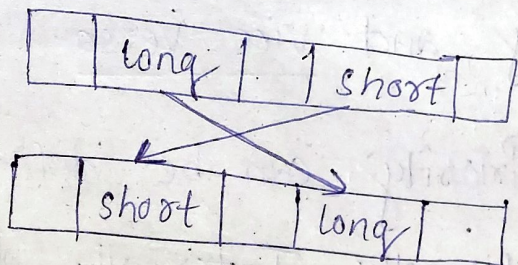
A priority scheduling algorithm can leave some low-priority process waiting indefinitely for the CPU.

- A solution for indefinite blocking is AGING.

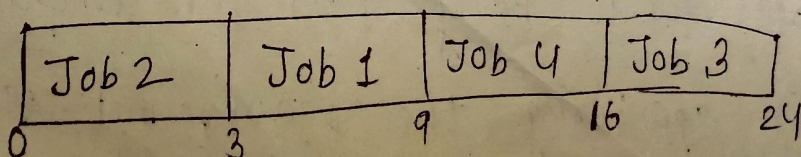
It is a technique which gradually increase the priority of jobs that waits in the system for a long time.

Consider the following set of jobs

<u>Job</u>	<u>Burst time</u>
1	6
2	3
3	8
4	7



Using SJF scheduling, the Gantt chart is



Hence the average turnaround time = $\frac{3+9+16+24}{4} = 13$

The real difficulty with SJF is knowing the length of the next CPU request. There is no way to know the length of the next CPU burst. We may not know the length of next CPU burst, but we may be able to predict its value.

The next CPU burst is generally predicted as an exponential average of the measured lengths of previous CPU burst. Let t_n be the length of the n^{th} CPU burst and let J_{n+1} be our predicted value for the next CPU burst.

Then for α , $0 \leq \alpha \leq 1$ define

$$J_{n+1} = \alpha t_n + (1-\alpha) J_n$$

This formula defines an exponential average.

" J_n " stores the past history.

P⁽¹⁰⁾ Preemptive Algorithms :

In preemptive scheduling a scheduling decision can be made even while the execution of a job is in progress. Consequently a job in execution may be forced to release the processor, so that execution of some other job can be undertaken i.e. the job of higher priority is served by extracting the lower priority job.

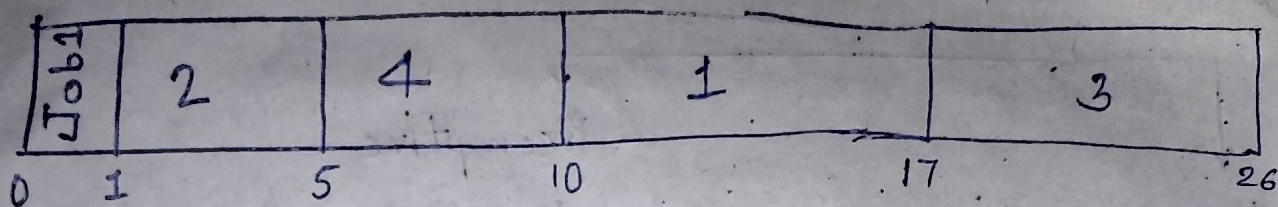
① Shortest job first (Preemptive)

Example :

<u>Job</u>	<u>Arrival Time</u>	<u>Processing time</u>
1	0	8
2	1	4
3	2	9
4	3	5

If job arrive at the ready queue at the times shown and need the indicate times, then the following Gantt Chart shows the

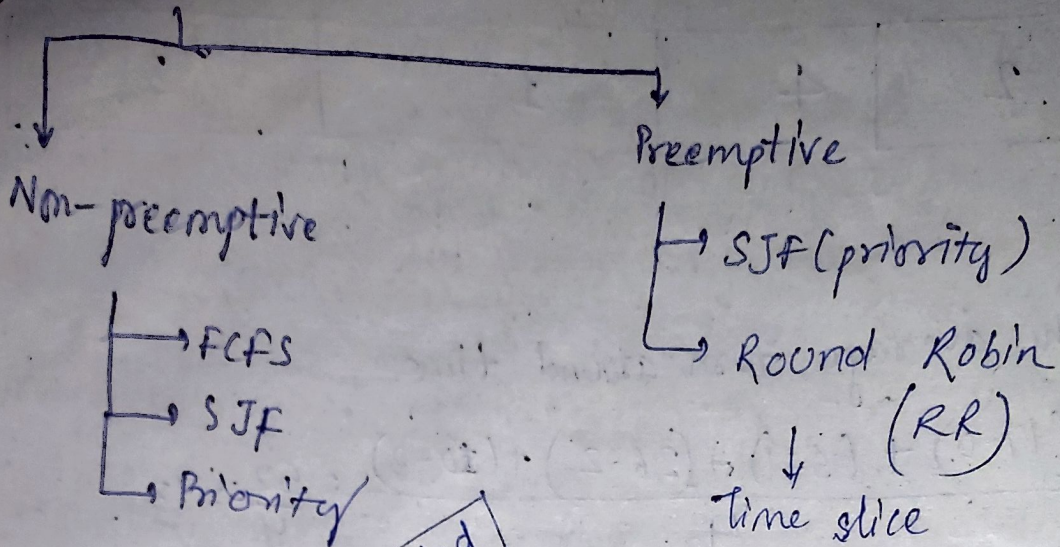
resulting preemptive SJF schedule



Here the average turn around time

$$= \frac{(17-0) + (5-1) + (26-2) + (10-3)}{4} = \frac{52}{4} = 13$$

29/01 Scheduling



Q (RR)

Process	CPU burst time
P ₁	10
P ₂	5
P ₃	2

adopted FCFS

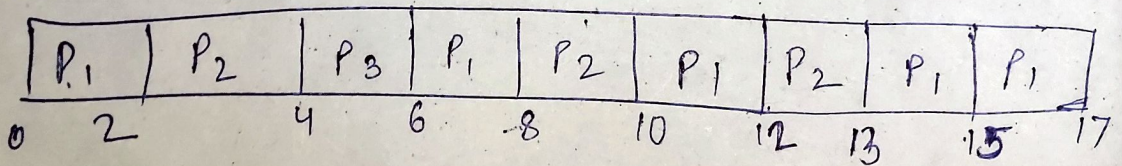
Priority	AT
3	0
2	1
1	2

(time quantum)

$t_q = 2 \text{ ms}$

It will return to wait state after 2ms execution of P₁.

Gantt chart



TAT

WT

$$\begin{aligned}
 17 - 0 &= 17 \\
 13 - 1 &= 12 \\
 6 - 2 &= 4
 \end{aligned}$$

$$\begin{aligned}
 4 + 2 + 1 &= 7 \\
 1 + 4 + 2 &= 7 \\
 2 + 0 &= 2
 \end{aligned}$$

$$\begin{aligned}
 \text{Avg. WT} &= \frac{7 + 7 + 2}{3} \\
 &= \frac{16}{3} = 5.33
 \end{aligned}$$

$$\text{Avg. TAT} = \frac{17 + 12 + 4}{3} = \frac{33}{3} = 11$$

Comparing both SJF (priority) & RR, it was found that SJF (priority) is better than RR.

If $AT = 0$, All jobs are arriving same time.

WT

$$1 + 2 + 1 = 7$$

$$2 + 4 + 2 = 8$$

$$4$$

~~TAT~~ TAT

$$17$$

$$13$$

$$6$$

Avg. TAT

$$= \frac{17 + 13 + 6}{3}$$

$$= \frac{36}{3} = 12$$

Avg. WT = $\frac{7 + 8 + 4}{3} = \frac{19}{3} = 6.33$

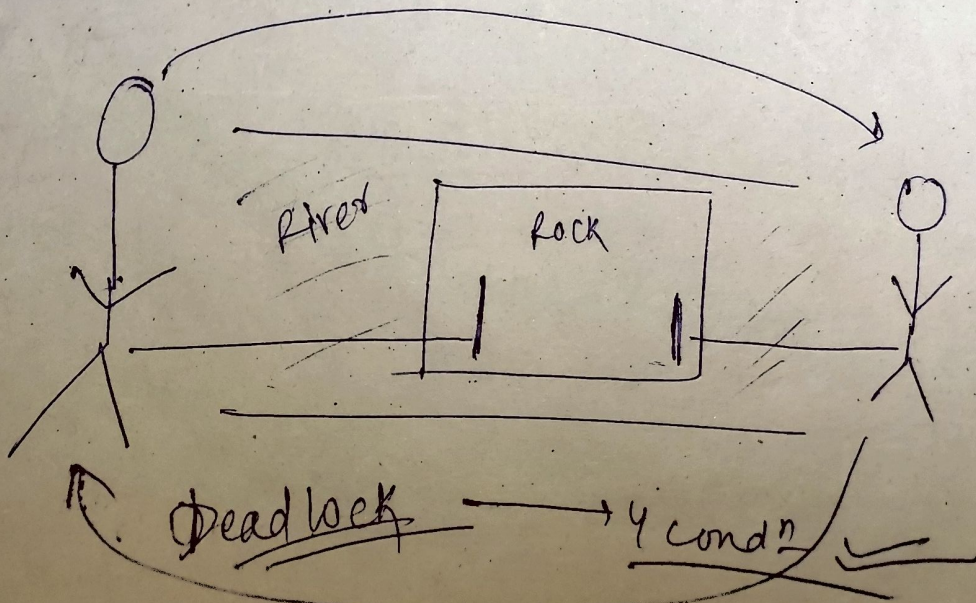
Dead Lock

Also ~~also~~ known as Deadly

When 2 process or more (P_0, P_1) unknowingly waiting for resources held by each other arises dead lock situation.

If 4 condns ^{occur} ~~are~~ fulfilled at a particular instant of time ~~a~~ only result in deadlock.

- 1. Mutual Exclusion
 - 2. Hold & wait
 - 3. No Preemption
 - 4. Circular wait
- No mutual among the processes. (Resources)
- Holding & waiting both process
- No forcefulness should occur
- Circularly waiting for each other



Overcoming deadlock → Dead lock recovering

2 condⁿ to overcome Dead lock:- → Th.
→ Practical.

- ① Deadlock avoidance → Once deadlock arises then
- ② Dead lock prevention to resolve it.

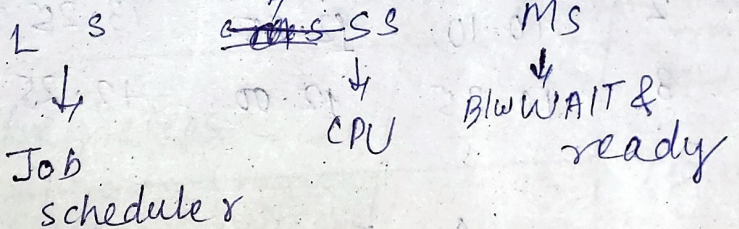
↳ Preventing before deadlock arise.

Banker's Algorithm → to resolve Deadlock.

30/01

DM
IM
MM
PM

- ① Process state
- ② Process scheduler



- ③ scheduling

Non-preemptive

Preemptive

- FCFS
- SJF
- Priority

- RR
- SJF (priority)

Q1

(Arrival Time) (Run Time)

Job scheduling

using FIFO (FCFS)

Job No.	AT	RT (hr)
1	10.00	2.00

2	10.10	1.00 hr
---	-------	---------

3	10.25	0.25
---	-------	------

Q2

Job No.

AT

Start Time

Finish Time

TAT

WT

1

10.00

10.00

12.00

2.00 hr

2

10.10

12.00

13.00

2.90 hr

3

10.25

13.00

13.25

3.00 hr

7.90 hr.

Avg. TAT = 2.63 hr.

Q2

Job scheduling using shortest Job First (SJF)

Q2

Job No.

AT

ST

FT

TAT

1

10.00

10.00

12.00

2.00 hr

2

10.10

12.25

13.25

3.15 hr

3

10.25

12.00

12.25

2.00 hr

7.15 hr.

Avg. TAT = 2.38 hr.

Q3

Job scheduling using future knowledge?

Job No.

AT

Start Time

FT

TAT = FT - AT

1

10.00

11.50

13.50

3.50

2

10.10

10.50

11.50

1.40

3

10.25

10.25

10.50

0.25 hr

5.15 hr

CPU idle = 0.25 hr

Avg. TAT = 1.7166 = 1.72 hr

$$AT - RT \rightarrow I$$

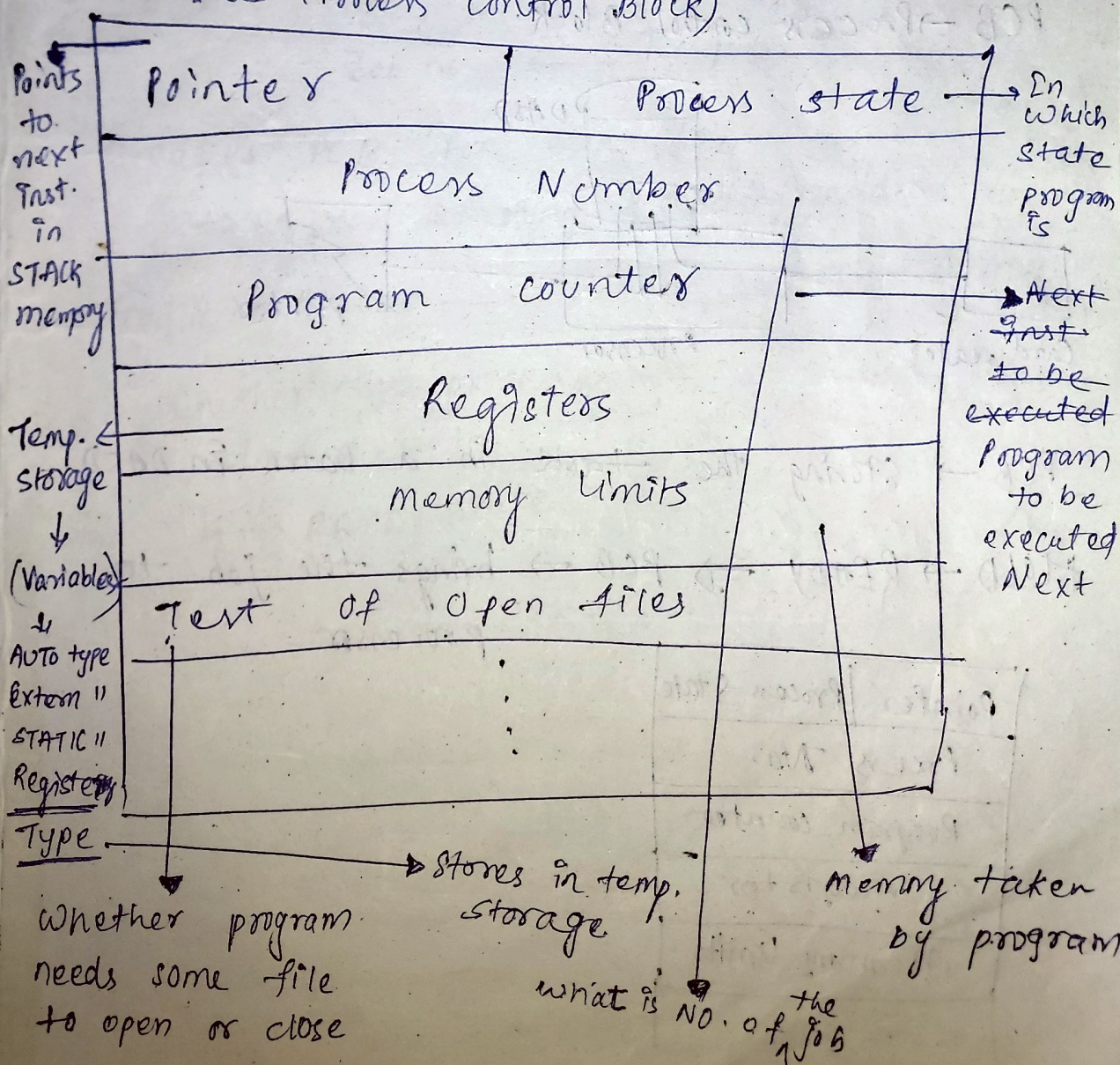
$| \text{Arrival time} + \text{Response time} | = \text{Waiting Time}$

future knowledge $\rightarrow$ such that TAT is less as compared to that of FCFS, SJF.

PCB / JCB / TCB $\rightarrow$ Process / Job / Task Control Block

$\rightarrow$ Acknowledges the priority of any

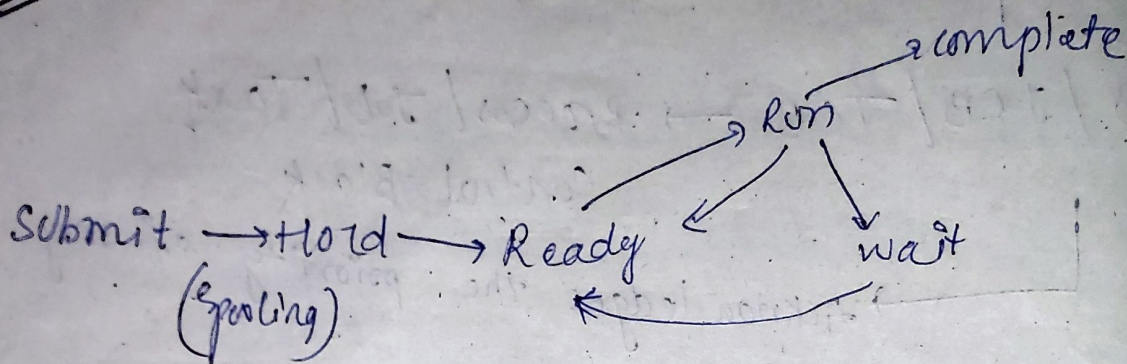
* PCB (Process control Block)



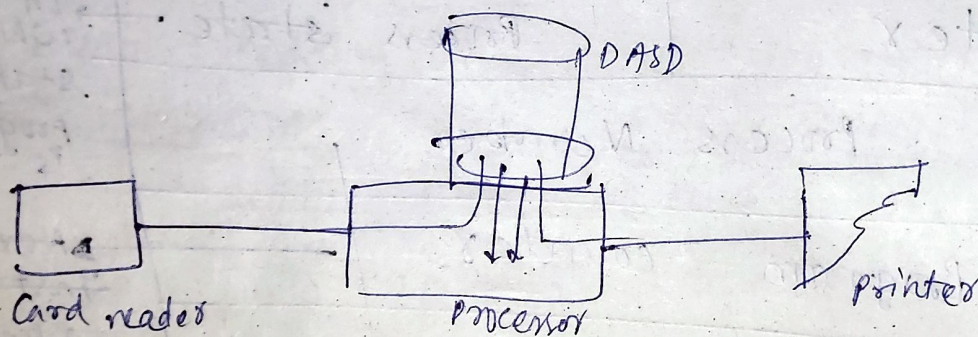
Synchronise ←

fopen (file name) ← Inst. to open file

31/01



PCB → Process control Block



PCB → storing the task in a batch in DASD

HOLD → READY ⇒ PCB → brings the job to processor.

Pointer	Process State
Process No.	
Program counter	
Register	
Memory limits	

OS → Keeps bird view ~~which~~ idea about job → How?

↓
which is
SJF, future
knowledge.

Base + off set ← starting
address

and end address

Processor → searches → starting address in
DASD. ←

HOLD → READY → Job no. ←

READY → RUN → stores in 1st memory.

Job no. ← Tag ←

Creates PCB for each job.

→ To know starting & ending address

Traffic controller → checks, WAIT → READY

whether the processor

is free or not

↳ READY state

② (PCB / JCB / TCB) Process / Job / Task control Block:

A PCB is a database or record containing many pieces of info. associated with a ~~specific~~ specific processes.

It includes: ① Process state → may be the NEW, READY, RUN ~~STATE~~, WAIT, etc.

① Program counter: PC indicates address of the next inst. to be executed for this process or job.

② The CPU registers vary in no. and type depending on CPU or ~~com~~ upon the computer architecture.

They include accumulator, index registers, and GPRS (General Purpose Registers)

③ MM → MM Info. include Base and Bound registers

HALT → Privileged inst. → forcefully stops the program.

④ Accounting Info. → Includes the amount of CPU and real time used

⑤ I/O status Info. → Includes outstanding I/O requests, and I/O devices allocated

to this processes. (13)

Buffering $\rightarrow$ offline.
computation of 1 program reqd is result of same program.

(vii) cpu scheduling Info - includes process priority, pointers to the scheduling queue.

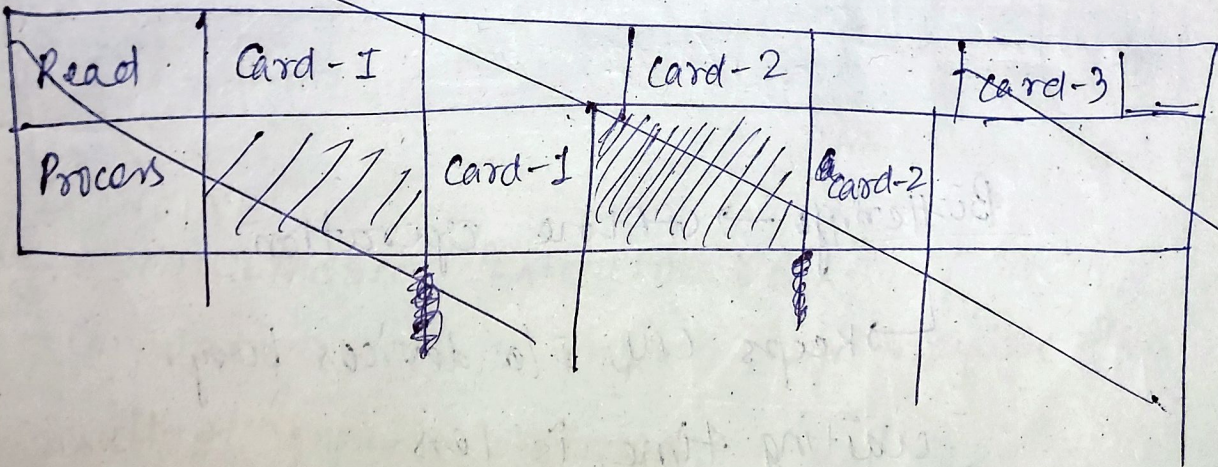
opt $\rightarrow$ Time complexity, memory ~~also~~ used

$\uparrow$ if less:
 $\rightarrow$ more priority.

Time space Trade off.

(viii) PCB serves as a repository for any info. that may vary from process to process.

Buffering $\rightarrow$



Buffering →

Read	Card-1		card-2		card-3	
Process		card-1		card-2		card-3
Print			card-1		card-2	

Without buffering → time

Here CPU or I/O device waits till one of either works complete.

Read	card-1	card-2	card-3	etc.		
Process		card-1	card-2	card-3	etc.	
Print			card-1	card-2	card-3	etc.

Buffering → offline operation.

↳ keeps CPU, I/O devices busy.
waiting time is less

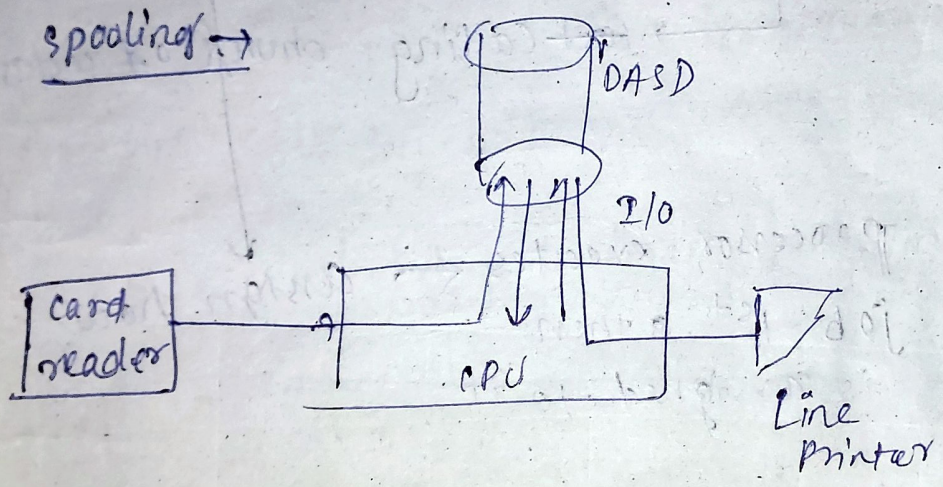
Either CPU or I/O devices complete first.

ASISD, CISC, CISC, MISD, MIMD
* SIMD,

05/02

short notes → Overlays
Time sharing
Spooling
Buffering
Multiprogramming
Precedence Inst.

spooling →



- ONLINE
- SIMULTANEOUS PERIPHERAL OPERATION ONLINE
- Computation of 1 program, o/p may be the result of another program.

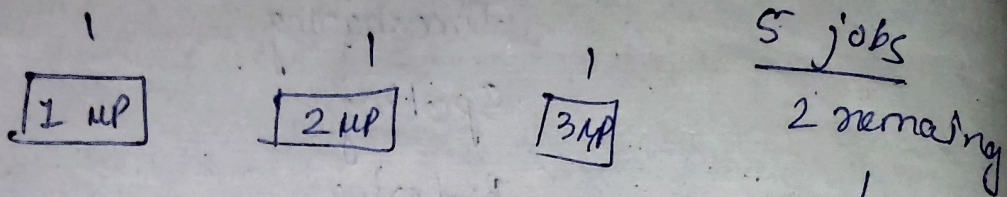
Buffering → OFFLINE

- Computation of 1 program, o/p is the result of the same program.

Time sharing →

1 processor various jobs
executing by time sharing

Multiprogramming → 1 processor and many jobs
executing one by one but so fast
that it seems concurrently.

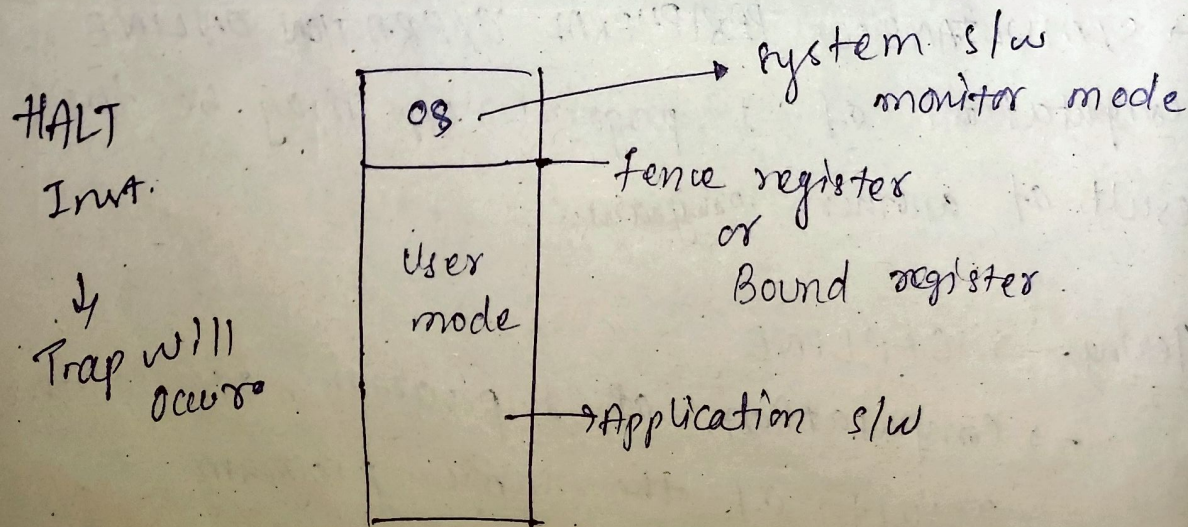


Remote Procedure Call → RPS

↳ ~~not~~ calling chunk of codes

which processor executes ← assign how?
current job 1st & then
another is assigned to it.

Privilege Inst. →



Trap → 8257

SISD $\rightarrow$ Single Inst. Single Data side
 SIMD $\rightarrow$ _____ multiple _____
 MIMD $\rightarrow$ multiple . _____ multiple -
 MISD $\rightarrow$ _____ single _____

Booting $\rightarrow$ starting the computer power button.

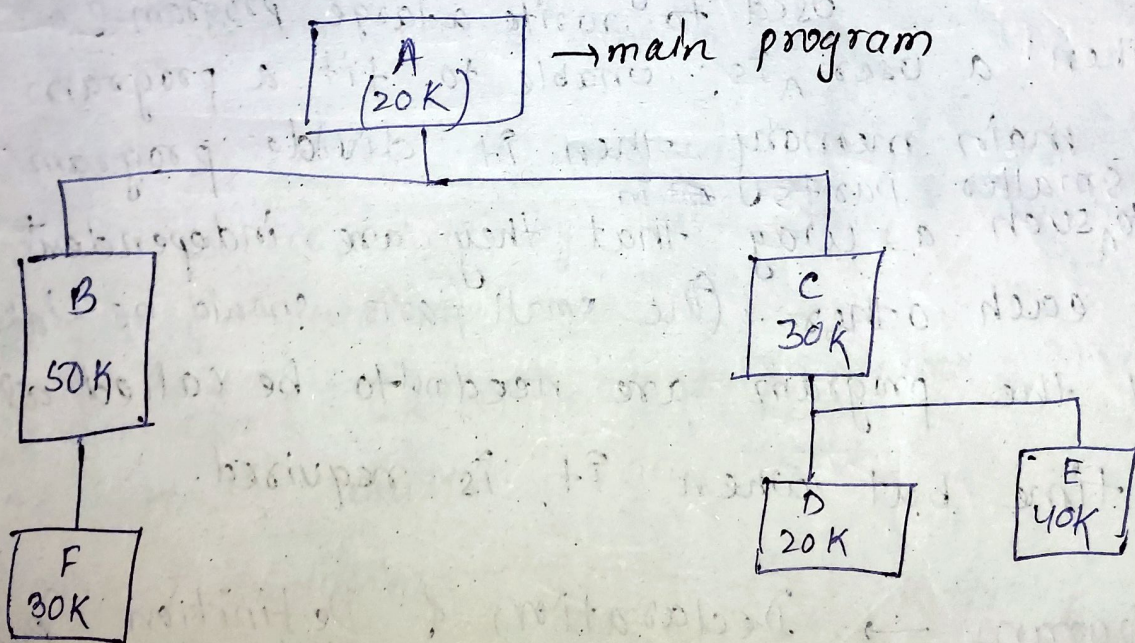
old called Booting $\rightarrow$ switch on ~~switch off~~

~~Hot start~~ Booting $\rightarrow$ Rebooting $\rightarrow$ system is allowed to restart when it goes to hang State

Overlay $\Rightarrow$

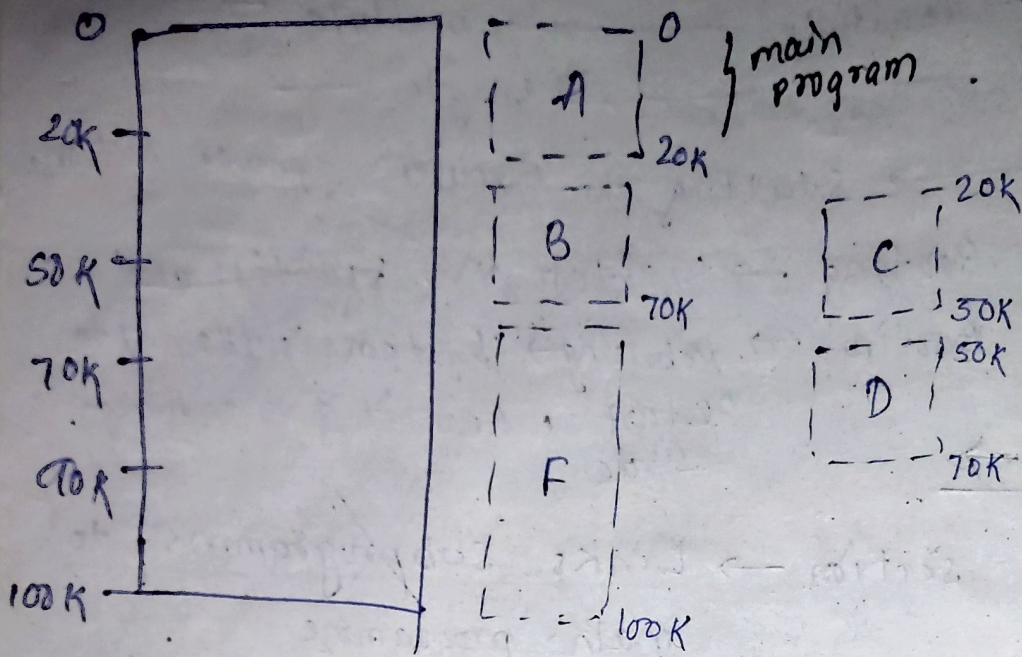
Linkage section $\rightarrow$ Links subprogramms. to main programme.

Procedure call structure



Total address space = 190 k bytes

Overlay memory management



Primary memory
used to

used to write a large program & when a user is unable to fit a program in main memory then it divide program into smaller parts ~~in~~ in such a way that they are independent of each other. (the small parts should be linked) all the program are need not to be called at a time but when it is required.

program → Declaration & Definition part

↓
variables are declared

↓
calling & using variable

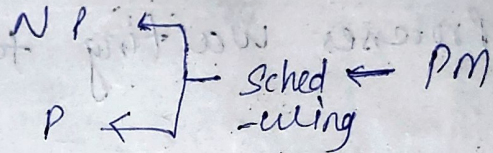
When A is called then it may either call B or C and then it will call its sub parts (which is required at that time) which will further be required

Process of dividing to small parts is known as overlays.

The small parts are known as overlays

Dead Lock (PM):

PM → MP
IM → data
DM → Device
MM → memory

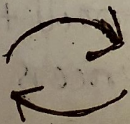


Deadlock
Synchronisation

When 2 or more processes unknowingly waiting for a resource held by each other arises dead lock situation.

If 4 condⁿ satisfy at a particular instant of time results in dead lock:

1. Mutual Exclusion → mutually excluded
2. Hold & WAIT → P₁ hold then P₂ wait and vice versa
3. No preemption → No forcefulness
4. Circular Wait → P₁ waiting P₂ to ~~not~~ release & vice versa



Dead Lock

Condⁿ :-

1. ~~At least~~ ^{At least} one resource is in non-sharable mode.
2. ~~At least~~ ^{Till} 1 process is holding a resource other process has to wait.
3. Acc. to willness only resource will be released no forefallness.
4. $\neq$ Processes waiting for each other.

Resource Allocation Graph :-

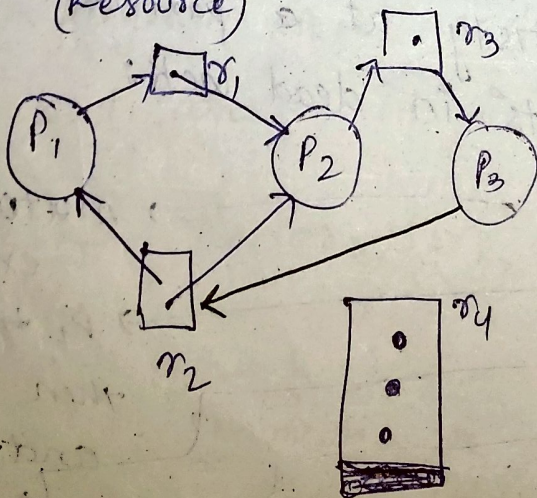
Graph $\rightarrow G = (V, E)$, $V \rightarrow$ Vertex
 $E \rightarrow$ Edges

corner pt
where 2 corner
pts meet

$\checkmark$
 P (Process)
 R (Resource)
Set of processes & set of resources

Resource

$\rightarrow$ card reader, printer



Arrow from Process $\rightarrow$ Request resource

Arrow to Process $\rightarrow$ Allocation

Processes $\rightarrow \{P_1, P_2, P_3\}$

resources $\rightarrow \{r_1, r_2, r_3, r_4\}$

$E \rightarrow \{(P_1, r_1), (P_2, r_3), (r_1, P_2), (r_2, P_1), (r_2, P_2), (r_3, P_3)\}$

one instance of resource type r_1
 two r_2
 three r_3
~~four~~ three r_4

cond? $\rightarrow$ Check if there is a cycle.

cycle $\Rightarrow$ Dead lock

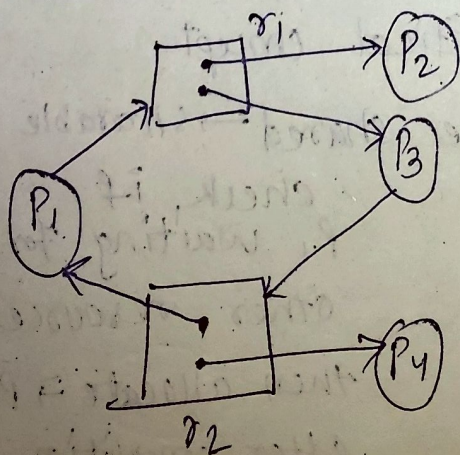
No cycle $\rightarrow$ No chance of dead lock

$(P_3, r_2) \rightarrow$ Dead lock $\rightarrow$ cycle $\Rightarrow (P_1, r_1, P_2, r_3, P_3, r_2, P_1)$

$P_1 \rightarrow r_1 \rightarrow P_2 \rightarrow r_3 \rightarrow P_3 \rightarrow r_2 \rightarrow P_1$

$r_2 \rightarrow P_2 \rightarrow r_3 \rightarrow P_3 \rightarrow r_2$

No dead lock with cycle



Processes $\rightarrow \{P_1, P_2, P_3, P_4\}$

Resources $\rightarrow \{r_1, r_2\}$

$E = \{(P_1, r_1), (P_3, r_2), (r_1, P_2), (r_1, P_3), (r_2, P_1), (r_2, P_4)\}$

cycle $\rightarrow (P_1, r_1, P_3, r_2) \rightarrow$ No dead lock

P_2 releases r_1 there is a chance / hope
and P_1 can be allocated

Also, there is a hope that P_4 may release
 r_2 and can be used by P_3 .

This is known as Live lock

~~two~~ instance of resource types,
 r_2

Handling Dead lock :-

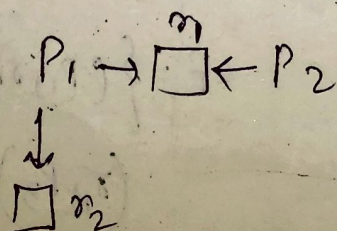
Two situations →

- ① Allow the system to go to dead lock and then recover → Dead Lock Avoidance
← Dijkstra's Algo. ← Also known as Banker's Algorithm
- ② Do certain thing so that system will never go to dead lock → Dead Lock Prevention

Dead lock Avoidance → Theoretical concept.

(mutual) → Resources can be shared → sharable

Hold (Hold & wait)



check if
 P_1 waiting for
other resources
then allocate $\Rightarrow P_2$
after completion
allocate to P_1

(preemption)
→ Forcefullness → One Process using resource 1st
and then other process will
use the resource
→ Use resource acc. to number in a sequence
(r_1, r_2, r_3) ↓
in a queue.

↓
No chance of Dead Lock

Card reader → 1
Punch Card → 2
mag. Tape → 3
mag. Disk → 4
Printer → 5

BUFFERING

Buffering is the name given to the technique of transferring data into temporary storage prior to processing (or) output, thus enabling the simultaneous operation of devices. This can be achieved because of the autonomous operation of peripherals which leaves the CPU free from other work.

→ It performs:

- (i) OFF-Line operation
- (ii) Computation of one program and o/p of the same program

Buffering keeps both the CPU and its I/O device busy all the time.

It means if the ~~computer~~ CPU is working on one record while an input device is working on another either the CPU (or) the input device will finish first. If the CPU finishes first, it must wait; it can't proceed until the next record is read, so CPU is idle for long time. If the input device finishes first, then either it must wait (or) it may proceed with reading the next record.

So if the buffering technique is used then the

input device can read several records ahead of the CPU. (15)

EX: Suppose 2000 cards need to be input and the details on the cards printed out on a line printer (one line being used for each card read). The card reader operates at 1000 cards per minute and the printer at 500 lines per minute. The processing cycle and the time required would be illustrated in fig. (A) if buffering were not used and as illustrated in fig. (B) if buffering were used.

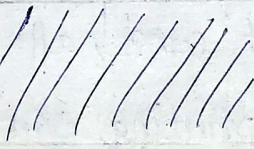
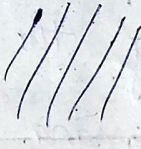
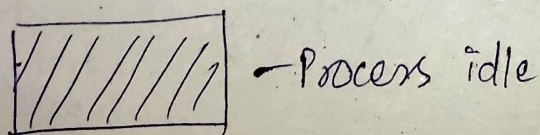
Read	card-1		Card-2		card-3	
Process		Card-1		Card-2		etc. ... Card-3
Print		Card-1		Card-2		

fig (A) Without buffering → time



Here one activity takes place at a time i.e. as the card is processed the reader and printer idle.

Total time required :-

$$\text{READ} = 2000 \times \frac{1}{1000} = 2 \text{ min}$$

PROCESS = negligible

$$\text{PRINT} = 2000 \times \frac{1}{500} = 4 \text{ minute}$$

$\therefore$ Total = 6 minutes

READ	Card-1	Card-2	Card-3	etc.
PROCESS	Card-1	Card-2	Card-3	etc.
PRINT		Card-1	Card-2	Card-3 etc.

fig (B): With Buffering $\rightarrow$ time

Total time:

After reading the first card, all subsequent reading, processing and printing can be overlapped. time required is a little over the speed of the device i.e. just over 4 minutes.

SPOOLING :-

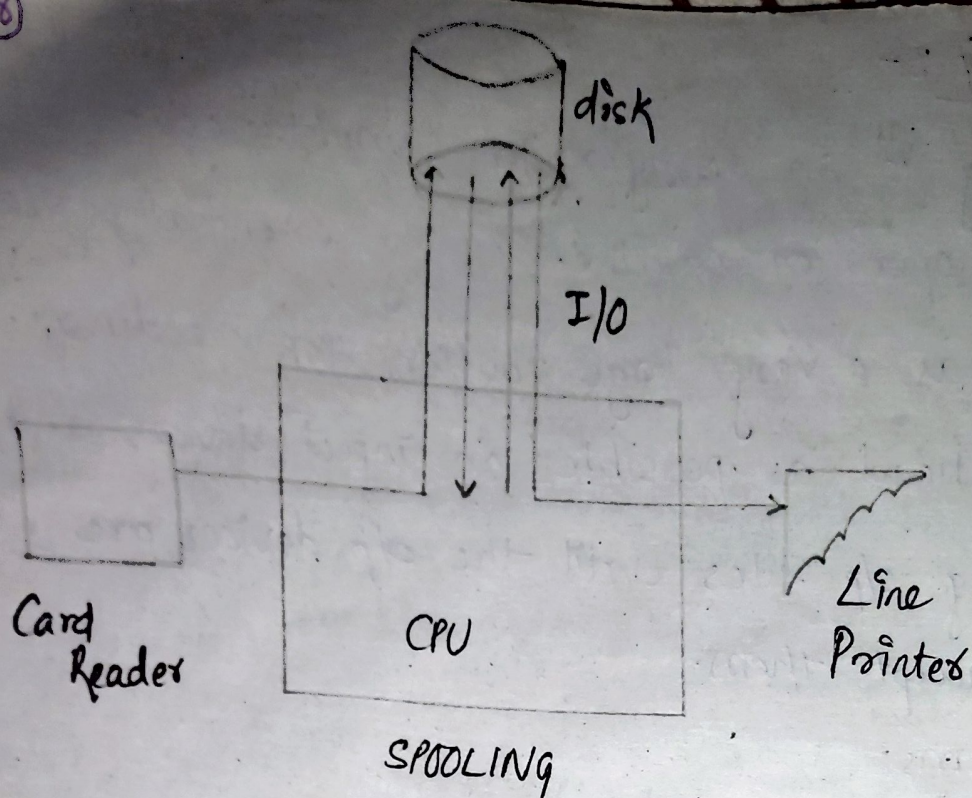
(17)

The spooling is an acronym for simultaneous peripheral operation on-line. Spooling essentially uses the disk as a very large buffer, for reading as far ahead as possible on input devices and for storing o/p files until the o/p devices are able to accept them.

It performs:

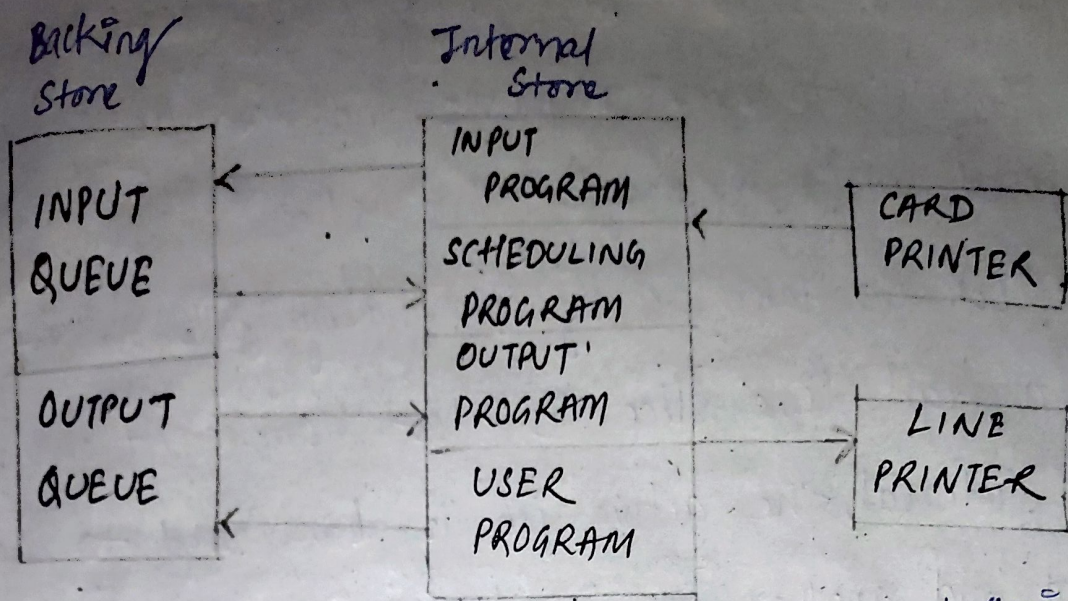
- (i) ON-Line operation
- (ii) Computation of one program and o/p may be the result of another program

Spooling can be done with magnetic tape. It is possible to read the entire contents of a magnetic tape onto disk before using it. All operation then occurs on the disk copy, at higher speeds and with no wear to the tape. This scheme is known as staging a tape.



Spooling can keep both the CPU and the I/O devices working at much higher rates, particularly if there is a mix of CPU-bound and I/O-bound jobs to be run.

* Ex: A very successful operating system with input/output spooling called Exec II, was designed by Computer Science Corporation (Lynch 1967 & 1971). It controlled a Univac 1107 computer with an instruction execution time of 4-sets. The backing store consisted of two (or) more fast drums, each capable of transferring 10,000 characters during a single revolution of 33 msec.



A spooling system controlling continuous buffering of input/output on backing storage and sequential scheduling of user jobs.

TIME SHARING SYSTEM

A Multiprogramming system that process many user programs concurrently in an interactive manner are called time sharing system. Here interactive manner means the computer and the terminal user may communicate with each other.

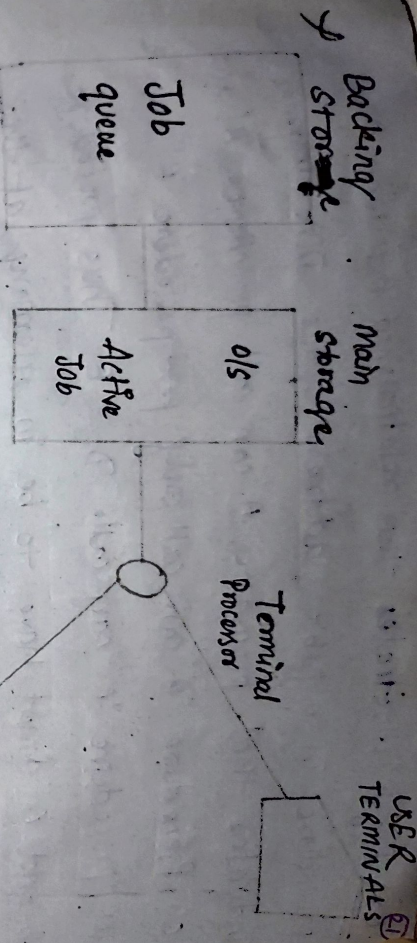
In such a system a clock is used to divide up processor time into "time-slice" and these time-slices are shared between users in an appropriate way. This is called time sharing. If a job is not completed at the end of its time-slice, then it is interrupted and returned to the end of a queue to wait for

Another "time-slice". This policy guarantees that fast response to user requests that can be processed with a single time slice.

Ex: let us take a system having following details. The average time slice is about 40 msec. At end of this interval, the active job is transferred to the queue and another job is loaded into the internal store. This exchange of jobs between two levels of storage is called swapping. It takes roughly 40 msec. 80 CPU needs 80ms for one job (i.e. one terminal).

Let us take that the system has 25 terminals. So a user can expect response to a single request with a single time slice only) in $25 * 80 \text{ msec} = 2 \text{ sec}$.

So here we see that the time sharing system provide fast response to user request.



An interacting system with low-speed user terminal and swapping of jobs between internal and backing storage.

OVERLAYS

When a program is used to write a large program that may not fit into the main memory. It is necessary to divide the program into small portions ~~are called~~ so that each one may be fit into the primary memory. These small portions are called overlays. A programmer has to design overlays so that they are independent of each other. Under these circumstances one can successively bring each overlay into the main memory and execute them in a sequence. But it increases the program-development time considerably.

(22)

Ex: Fig. 0 illustrates the relationship between the procedure in a job's address space. This diagram indicates that procedure 'A' can call procedures 'B' and 'C'. Procedure 'B' can call only ~~procedure~~ procedure 'F', where as procedure 'C' can call ^{group} D & E. Thus procedure B and C donot have to be in memory at the same time. So instead using 190K bytes to hold the job's address space in memory, maximum of 100K bytes are needed.

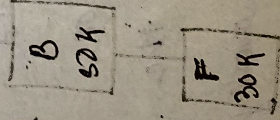
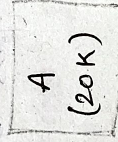
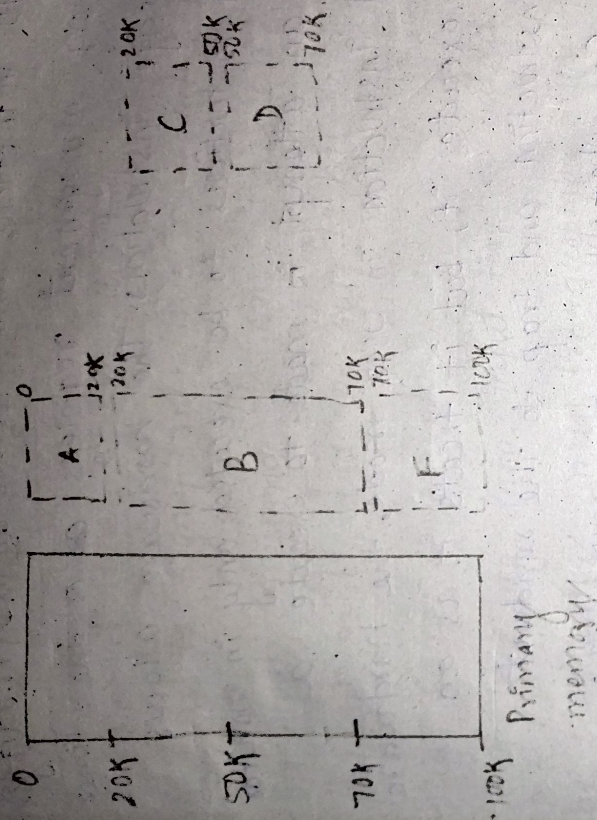


Fig. ①

Total address space = 190K bytes.

a) Procedure call structure



⑥ Overlay memory management

PRIVILEGED INSTRUCTIONS

Privileged instructions is a special type of instructions which executed only in monitor mode.

Operating system can execute privileged instructions as it executes in monitor mode. So it change the system state from user to monitor mode.

To provide better service, we have two separate mode of operation:

(i) User mode

and (ii) monitor mode (also called system mode)

A bit is added to the hardware of the computer to indicate the current mode i.e. user (1), monitor (0).

(29)

I/O instructions and instruction to modify the memory management registers ~~as~~ or the times are privileged instructions. The hardware allows privileged instructions to be executed only in monitor mode. If an attempt is made to execute a privileged instruction in user mode, the hardware will not execute it, but it treats it as an illegal instruction and trap to the resident monitor. So ~~when~~ when a trap occurs, the hardware switches from user mode to monitor mode.

EX. OF PRIVILEGED INST;

The HALT instruction is privileged, as a user program should never be able to halt the computer. The instructions to turn the interrupt system on and off are also privileged.

REAL TIME SYSTEM

(25)

In a real time system, the time at which output produced is significant. This is usually because the input correspond to some movement in the physical world and the output has to relate to that same movement.

The lag from input time to output time must be sufficiently small for acceptable timeliness.

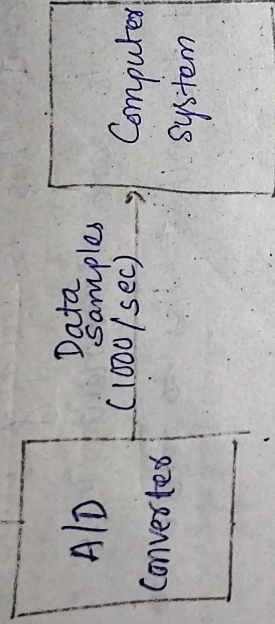
Timeliness is a function of the total system.

The main advantages of real time system is that the response time is of order milliseconds.

Ex: In the real time system, digitised samples are generated at the rate of 1000 samples per second.

The computer is only required to store these samples in a specified time. Since a new sample comes in every milliseconds, the computer must respond to every "store the sample" request in less than 1 ms.

If it fails to do so, a sample would be lost and the real time application would have malfunctioned. The worst case response time is thus 1 ms.



Real time application

Since the response time requirements of a real time system are critical, the real time application would be given higher processing priority.

NOTE

- ① In a real time system the response time is of order millisec.
- ② In an interactive system the response time is of order sec.
- ③ In a batch system the response time is of order hour.

Batch Processing

(27)

Batch Processing is execution of a series of programs ("jobs") on a computer without manual intervention. Jobs are set up so they can be run to completion without manual intervention. So, all input data are preselected through scripts, command-line parameters, or job control language. This is in contrast to "online" or interactive programs which prompt the user for such input. A program takes a set of data files as input, processes the data, and produces a set of output data files. This operating environment is termed as "batch processing" because the input data are collected into batches of files and are processed in batches by the program.

07/10/2

Two situations

①

Dead lock Prevention

②

Avoidance

→ Allow the system to go to dead lock

↳ (Banker's Algo.)

↳ to go the dead lock then recover

Recovering is hard

once system is in dead lock

(very costly)

Circular wait is prime condⁿ

↳ if true then any chance of dead lock.

→ Mutual Exclusion

③

Handling of Dead locks →

In order to handle dead lock problem we may use 2 process:

① Allow the system to enter a dead lock state & then recover.

② Doing some jobs that we never get into dead lock problem / state.

By allowing the system to be in deadlock and ⁽²⁹⁾ then recover is quite difficult & expensive there fore the 2nd method is acceptable.

There are 2 common methods:

① Dead lock Prevention

② Avoidance or Banker's Algorithm

Dead lock prevention:-

We know there are 4 cond^s to have deadlock: But if one of them will not occur/held the deadlock problem can be avoided.

Mutual Exclusion: The mutual exclusion condⁿ holds only for non-sharable device.

For example printer can not be shared among several processes at a time or similar time.

(Inter-mixing o/p $\rightarrow$ Race condⁿ)
critical section

But if we have a read only file then it can be shared among several processes i.e. several processes can open a read only file simultaneously. So a process never have to wait for a sharable device.

Eg: Railway reservation system

At the same time all every job can access the database.

Disadv: Some resources are non-sharable
↳ Printer

Hold & wait: (Pre-allocate all shared resources)

In order to ensure that the hold & wait condⁿ never halts in the system there are 2 ways:

① All the resources that are assumed to be used by the process, must ~~be~~ assigned to that process before the beginning of execution.

② A process ~~must~~ ^{may} request some resource & when it has none and use the resource before it can request any additional resources. However it must release the resources that ~~it~~ is currently allocated so that there will be no hold & wait condⁿ.

Disadv: In the 1st case resource utilization may very slow since many resources ~~can~~ ^{may} be allocated but not used for a long period.

In the 2nd case we have to ~~not~~ ensure that our data ⁽³¹⁾ will remain on the allocated resources or not.

There may be starvation for popular resources. It will wait till all the resources are available.

No - Preemption: In order to ensure that

no-preemption never haults there are 2 ways:

① If a process is holding some resources and requesting another resource then all the resources of the process should be released and allocate the resources to another process, which is waiting for, by putting the ^{1st} process in Block State.

The process will only be restarted when it can regain ~~at~~ its old resources as well as new one which it is waiting for.

② If a process request some resources, we ^{1st} check if they are available, if so allocate them. If they are not available, will check whether they are allocated to some other process which is waiting for some additional resources. if so preempt the desire resources

from the waiting processes and allocate them to the request-process.

The waiting process is only be^{re}started when it is allocated new resources it is requesting for.

Circular Wait: In order to ensure that circular wait never happens then assign a unique integer no. to all resource & request the resource as per the order which is a legal sequence.

(card reader $\rightarrow$ 1
Punch Card $\rightarrow$ 2
mag. Tape $\rightarrow$ 3
mag. Disk $\rightarrow$ 4
Printer $\rightarrow$ 5 etc.

But if request card reader 1, printer 5 and punch card 2 ~~then it is~~ which is not a legal request. In this way we can prevent dead lock.

Deadlock Avoidance (Banker's Algo)

Step 1 $\rightarrow$ Request $\leq$ need $\longrightarrow$ if ~~not~~ request $>$ need
it is illegal.

Step 1 $\rightarrow$ if satisfies then only proceed to further steps.

Step 2 $\rightarrow$ Request $\leq$ Available.

Step 3 $\rightarrow$ (i) available $:=$ available - request

(ii) allocation $:=$ ~~available~~ + request

(iii) need $:=$ need - request

Check above cond^{ns}.

If in a balanced way then no deadlock.

Ex: One type of resource $\rightarrow$ Printer

Process	Need	Request	Available	Allocation
P ₀	10	5	3	5
P ₁	<u>4</u> ✓	$4 \leftarrow 2 + \textcircled{2}$	$\leftarrow \textcircled{1}$	<u>$2 + 2 = 4$</u> ✓
P ₂	<u>9</u>	$4 \leftarrow 2 + \textcircled{2}$	$\leftarrow \textcircled{1}$	<u>$2 + 2 = 4$</u>

P₀, P₁, P₂ $\rightarrow$ Step-1 ✓

Step-2 $\rightarrow$ P₁, P₂ ✓ P₀ ✗

Step-3 $\rightarrow$ (i) ✓ (ii) ✓ (iii) P₁ $4 - 2 = 2$ ✓
P₂ $9 - 2 = 7$

P₁ $\rightarrow$ allocate $\rightarrow$ reclaim $\rightarrow 4 + 1 = 5 \rightarrow$ Algo. for P₀ & P₂.

Step 1 $\rightarrow P_0 \checkmark, P_2 \checkmark$ Available = 5

Step 2 $\rightarrow P_0 \checkmark, P_2 \checkmark$

Step 3 $\rightarrow$ (i) $P_0 \rightarrow 0$ (ii) $P_0 \rightarrow 10$ (iii) $P_0 \rightarrow 5 \checkmark$
 $P_2 \rightarrow 3$ $P_2 \rightarrow 4$ $P_2 \rightarrow 7$

Allocate $\rightarrow P_0 \rightarrow$ reclaim $\rightarrow$ Allocate P_2
when available = 10

① $\rightarrow 0 \rightarrow 10$

② $\rightarrow 0 \rightarrow 10$

DEADLOCK (OR) DEADLY EMBRANCE

(33)

Definition—

- ① A deadlock (or) deadly embrace is a situation in which two processes are unknowingly waiting for resources that are held by each other and thus unavailable.
- ② When two (or) more processes unknowingly waiting for an event to happen (which will never get happened) then the system is said to be in deadlock.

CONDITION:

A ~~deadly~~ deadlock situation can arise if the following four conditions held simultaneously in a system.

- ① Mutual exclusion
- ② Hold and wait
- ③ No preemption
- ④ Circular wait

(i) MUTUAL EXCLUSION:

At least one resource is held in a non-sharable mode, i.e. only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the

Resource has been released.

(II) HOLD AND WAIT:

There must exist a process that must be holding at least one resource and is waiting to acquire additional resource that are currently being held by other processes.

(III) No Preemption:

Resources can be not be preempted i.e. a resource can only be released voluntarily by the process holding it, after the process has completed its task.

(IV) CIRCULAR WAIT:

There must exist a set $\{P_0, P_1, \dots, P_n\}$ of waiting processes such that ' P_0 ' is waiting for a resource which is held by ' P_1 ' and ' P_1 ' is waiting for a resource which is held by $P_2 \dots P_{n-1}$. P_{n-1} is waiting for a resource which is held by P_n and P_n is waiting for a resource which is held by P_0 .

All these four condition can take place at a time.

Ex: Let us assume two people crossing the river (5)
from opposite sides meet at the middle, and
there is one way to cross the river which
can be used by one person at a time.
person can be stepping on a stone at a time.

The hold and wait condition is satisfied
since each person is stepping at one stone and
waiting to step to next one.

The no-preemption condition is hold since
stepping on the stone can't be forcibly removed
from the person stepping on it.
The circular wait condition holds since the person
coming from the opposite sides are waiting for
each other.

Resource Allocation Graph.

Deadlock can be described more precisely in terms
of a directed graph called system resource
allocation graph. This graph consists of a pair
 $G = (V, E)$ where 'V' is the set of vertices and 'E'
is the set of edges. The set of vertices is divided
into two types, 'P' consists of all processes and
'R' consists of all resources.

35) Let us take an example where
* The sets P, R and E

$$P = \{P_1, P_2, P_3\}$$

$$R = \{r_1, r_2, r_3, r_4\}$$

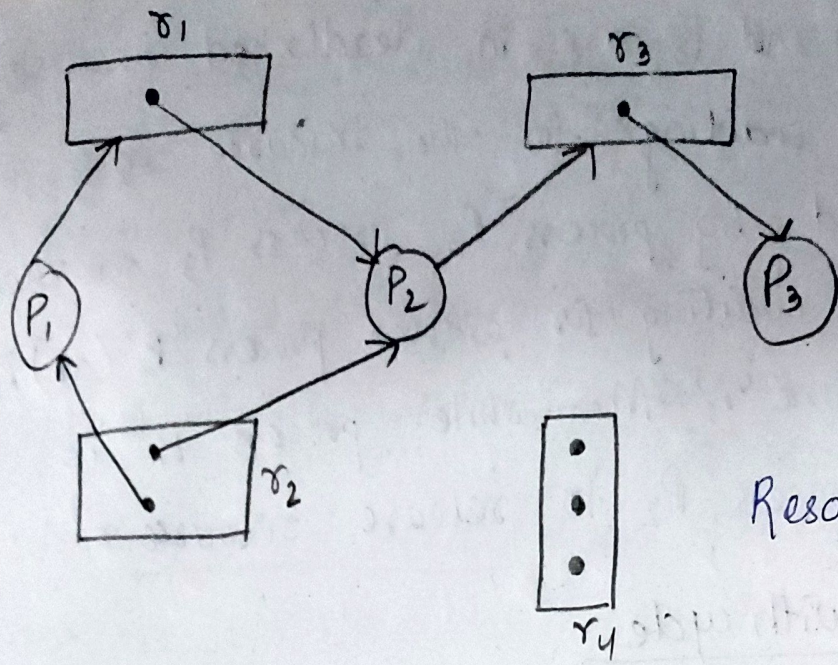
$$E = \{(P_1, r_1), (P_2, r_3), (r_1, P_2), (r_2, P_2), \\ (r_2, P_1), (r_3, P_3)\}$$

Resource instances:

- * one instance of resource type r_1 .
- * Two instances of resource type r_2 .
- * One instance of resource type r_3 .
- * Three instances of resource type r_4 .

Process status:

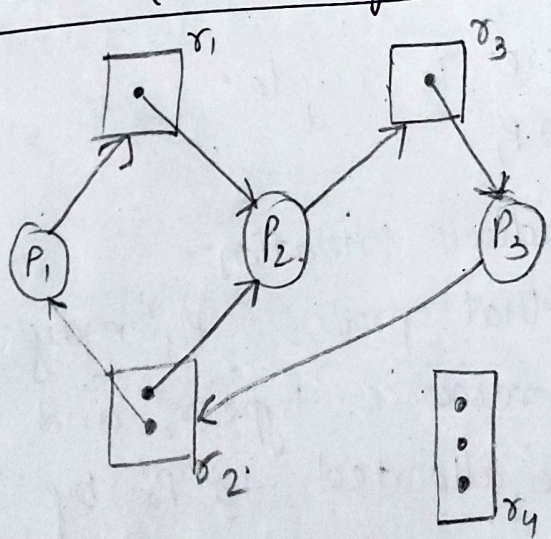
- Process P_1 is holding an instance of resource type r_2 and is waiting for an instance of ~~resource~~ resource type r_1 .
- Process P_2 is holding an instance of r_1 and r_2 and is waiting for an instance of resource type r_3 .
- Process P_3 is holding an instance of r_3 .



Resource Allocation Graph

In the fig. there is no cycle and here no dead lock occurs. Cycle is not a sufficient condition to have deadlock. But if there is a cycle then there may exist a deadlock (or) it may not be.

Deadlock With Cycle



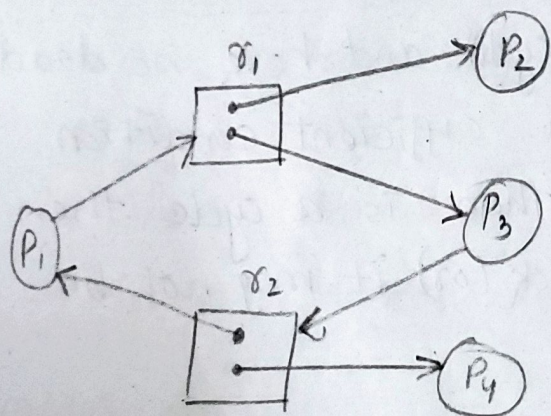
Resource allocation graph with a deadlock
Here, in this figure there are two cycles.

$$P_1 \rightarrow r_1 \rightarrow P_2 \rightarrow r_3 \rightarrow P_3 \rightarrow r_2 \rightarrow P_1$$

$$P_2 \rightarrow r_3 \rightarrow P_3 \rightarrow r_2 \rightarrow P_2$$

38) Process P_1 , P_2 and P_3 are in deadlocked because process P_2 is waiting for the resource r_3 , which is held by process P_3 process P_3 on the other hand is waiting for either process ' P_1 ' (or) P_2 to release resource ' r_2 '. Meanwhile process ' P_1 ' is waiting for process P_2 to release resource ' r_1 '.

No Deadlock with cycle



Resource allocation graph with cycle but no deadlock.

Here in this figure there is a cycle.

$P_1 \rightarrow r_1 \rightarrow P_3 \rightarrow r_2 \rightarrow P_4 \rightarrow P_1$

But here there is no deadlock situation.

Because there is a hope that process ' P_4 ' may release its instance of resource type ' r_2 ' and that resource can then be allocated to ' P_3 ' by breaking the cycle.

12/02/24 Banker's Algorithm:

This algorithm is given by Dijkstra.

When a ~~resource~~ request for resources is made by process P_i , the following actions are taken:

- 1) If request $\leq$ need, then proceed to step 2 otherwise we have an error, since the process has exceeded its maximum ~~reclaim~~.
- 2) If request $\leq$ available then proceed to step 3 otherwise the resources are not available and P_i must wait.
- 3) The system pretends to have allocated the requested resources to process P_i by modifying the state as below:

$$\text{Available} = \text{Available} - \text{Request}$$

$$\text{Allocation} = \text{Allocation} + \text{Request}$$

$$\text{Need} = \text{Need} - \text{Request}$$

If the resulting resource allocation state is safe the transaction is completed and process P_i is allocated its resource.

However if the new state is unsafe, then P_i must wait for ~~resource~~ request; and the old resource allocation state is restored.

<u>Process</u>	<u>Need</u>	<u>Request</u>	<u>Available</u>	<u>Allocation</u>
P ₀	10	5	3	5
P ₁	4	2		2
P ₂	9	2		2

Step 1 request $\leq$ need $\rightarrow P_0, P_1, P_2 \checkmark$

Step 2 request $\leq$ available $P_0 \times ; P_1, P_2 \checkmark$

Step 3 (i) available = available - request

$$P_1 \rightarrow 3 - 2 = 1$$

$$P_2 \rightarrow 3 - 2 = 1$$

(ii) allocation = allocation + request

$$P_1 \rightarrow 2 + 2 = 4$$

$$P_2 \rightarrow 2 + 2 = 4$$

(iii) Need = Need - request

$$P_1 \rightarrow 4 - 2 = 2 \quad \checkmark$$

$$P_2 \rightarrow 9 - 2 = 7 \quad \times$$

P₁ is allocated the resources 2 & 1 is remaining. Resources from P₁ were reclaimed.

$$\text{available} = 1 + 4 = 5$$

Now, P₀ & P₂.

Step 1 $\rightarrow P_0 \& P_2$ ✓

5 ← Available

Step 2 $\rightarrow$ ✓ $P_0 \& P_2$ ✓

Step 3 $\rightarrow$ (i) available = available - request

$$P_0 \rightarrow 5 - 5 = 0 \quad \checkmark$$

$$P_2 \rightarrow 5 - 2 = 3 \quad \checkmark$$

(ii) allocation = allocation + request

$$P_0 \rightarrow 5 + 5 = 10 \quad \checkmark$$

$$P_2 \rightarrow 2 + 2 = 4 \quad \checkmark$$

(iii) Need = Need - Request

$$P_0 \rightarrow 5 - 5 = 0 \quad \checkmark$$

$$P_2 \rightarrow 5 - 2 = 3$$

P_0 will be allocated to P_0 & then reclaimed
will be next allocated to P_2 when
available = 10.

$P_1 \rightarrow P_0 \rightarrow P_2 \Rightarrow$ step state

Q. To illustrate the banker's algorithm
consider a system with 5 processes P_0, P_1, P_2, P_3, P_4 and 3 resources A, B, C. Resources
type A has 10 instances, Resource type B
has 5 instances & resource C has 7 instances.
Suppose at time the following snapshot
of system is taken.

Process	Allocation			(Need) Max		
	R ₁	R ₂	R ₃	R ₁	R ₂	R ₃
P ₀	0	1	0	7	5	3
P ₁	2	0	0	3	2	2
P ₂	3	0	2	9	0	2
P ₃	2	1	1	2	2	2
P ₄	0	0	2	4	3	3

Available
~~A~~ → R₁ = 10
 B → R₂ = 5
 C → R₃ = 7

 R₁ → 10 - 7 = 3
 R₂ → 5 - 2 = 3
 R₃ → 7 - 5 = 2

Step 1: request & need R₁

R₁ → P₀ & P₂ & ~~P₁~~ & ~~P₄~~
P₁ & P₃ ✓

R₂ → P₀ & P₁ ✓, ~~P₂~~, P₃ ✓, P₄ ✓
P₂ ✓

R₃ → P₀ & P₁ ✓, P₂ ✓, P₃ ✓, P₄ ✓

the P₁ → P₃ available matrix is (3, 3, 2), Now we have to calculate the need matrix

Process	Need = Max - Allocation		
	R ₁	R ₂	R ₃
P ₀	(7-0) 7	(5-1) 4	(3-0) 3
P ₁	(3-2) 1	(2-0) 2	(2-0) 2
P ₂	(9-3) 6	(0-0) 0	(2-2) 0
P ₃	(2-2) 0	(2-1) 1	(2-1) 1
P ₄	(4-0) 4	(3-0) 3	(3-2) 1

P₁ → P₃

P_1 & P_3 satisfies & now P_1 is allocated 1st acc. to FCFS and then P_3 is allocated.

Now we know that need matrix & available matrix, so we can apply the banker's algorithm.

In this example, search the need matrix from P_0 to P_4 which is less than the available matrix $(1, 2, 2) < (3, 3, 2)$. Then allocate the requesting resource to P_1 . Now the process P_1 allocation is $(2, 0, 0) + (1, 2, 2) = (3, 2, 2)$.

After completion of job P_1 , it releases the total resource, the available resource is $(3, 2, 2) + (2, 0, 0) = (5, 3, 2)$.

Search the need matrix again, which is less than available $(0, 1, 1) < (5, 3, 2)$ so

allocate the resources to P_3 . After the completion of P_3 , it releases all its resources. Then the available = available + allocation i.e. $(5, 3, 2) + (2, 1, 1) = (7, 4, 3)$.

Continue this process until we find the safe state otherwise wait. The safe sequence for the problem.

$P_1 \rightarrow P_3 \rightarrow P_0 \rightarrow P_2 \rightarrow P_4$.

Resource $\rightarrow r_1 = 2+3 = 5$

$r_2 = 2+1 = 3$

$r_3 = 2$

After P_1

$(5, 3, 2)$

Process	Need = max - Allocation		
	R_1	R_2	R_3
P_0	$(4-0) \quad 4$	$(5-1) \quad 4$	$(3-0) \quad 3$
P_2	$(4-3) \quad 1$	$(0-0) \quad 0$	$(2-2) \quad 0$
P_3	$(2-2) \quad 0$	$(2-1) \quad 1$	$(2-1) \quad 1 \rightarrow V$
P_4	$(4-0) \quad 4$	$(3-0) \quad 3$	$(3-2) \quad 1 \rightarrow \text{excess}$

res ~~P_0~~ P_3 is allocated.

Free resources $\rightarrow$

$r_1 = 7$

$r_2 = 4$

$r_3 = 3$

After P_3

Process	Need = Max - Allocation		
	R_1	R_2	R_3
P_0	(7-0) 7	(5-1) 4	(3-0) 3 → ✓
P_2	(9-3) 6	(8-0) 8	(2-2) 0 → ✓
P_4	(4-0) 4	(3-0) 3	3-2 1 → ✓

P_0 is allocated

After allocation,

available resources

$$r_1 = 7, r_2 = 5, r_3 = 3$$

$$P_0 \rightarrow P_2 \rightarrow P_4$$

$$P_1 \rightarrow P_3 \rightarrow P_0 \rightarrow P_2 \rightarrow P_4$$

13/02/24.

Synchronization :-

Sitting barbar Problem

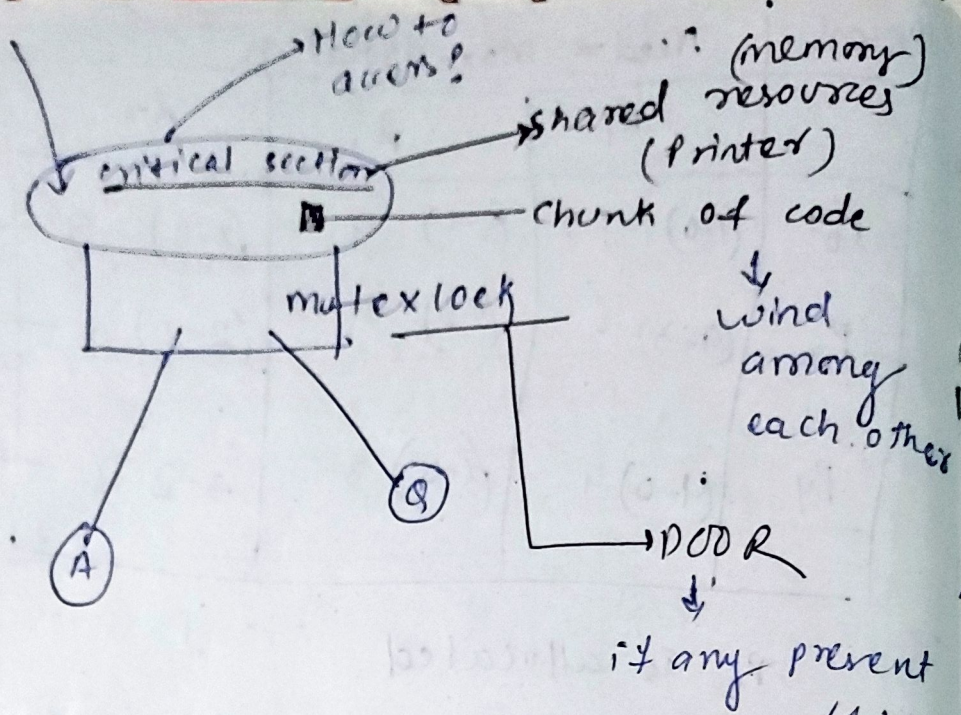
Dining Philosopher Problem

Process should be in ALERT

State to receive info.

and act

immediately to be in sync.



Railway Reservation System

How to access → At a time only one process can access
 → PCs

Mutex lock → DODR → 1 → Not empty
 0 → Empty

- To check
 - Lock
 - Semaphore
 - P & V
 - test & set
 - up & down
 - wait & signal
- } also known as

Synchronization - 2

When there are so many process in a system i.e. in a multiprogramming mode there is a need to share memory or resources b/w the processes. One the way of processing their arises certain unusual problems like a ~~process~~ process may be ^{requesting for} accessing a shared memory which is currently being used by another process.

→ If the shared memory is assigned to that process, accessing it (~~req~~ which is requesting for) without terminating the 1st one then the op will be a intermixed one.

This type of condⁿ is called as race condⁿ and that part of the program where the shared memory is accessed is called critical section.

A solⁿ to the critical section problem must satisfy the following 4 requirements:

① Mutual Exclusion: If process P_i is executing in its critical section then no other process ^{can be} ~~is~~ executing in its critical section. (No ~~to~~ ~~can~~ two contending processes should

~~Must~~ be executing simultaneously inside their critical section).

(i) Progress: If no process is executing in its critical section and there exist some processes that they wish to enter their critical section, then only those processes that are not executing in their remainder section can participate in the decision (As to who will enter the critical section next and this selection cannot be postponed indefinitely).
* (Process executing in remainder section is not considered but waiting processes are considered)

(ii) Bounded waiting: There must exist a bound on the no. of times so that other processes will be allowed to enter their critical section after a process has made a request to enter its critical section & before that request is granted. (It means no process should have to wait forever to enter its critical section)

(10) No error assumption : No assumptions should be made about relative speed and priorities of contesting processes.

(No priority & no relative speed)

Time	Process State			mutex 1 = free 0 = busy	CS attempt to enter
	P ₁	P ₂	P ₃		
M ₁	—	—	—	1	
M ₂	wait(s)	wait(s)	wait(s)	1	— : P ₁ , P ₂ , P ₃
M ₃	C.S	wait	wait	0	P ₁ : P ₂ , P ₃
M ₄	signal(s)	wait	wait	1	— : P ₂ , P ₃
M ₅	remainder	C.S	wait	0	P ₂ : P ₃
M ₆	wait	signal(s)	wait	1	: P ₃
M ₇	wait	remainder	CS	0	P ₃

wait(s) → semaphore

Semaphore :-

In multitasking systems, a semaphore is a variable with a value that indicates the status of a common resource.

→ It is ^{required to} ~~used to~~ lock the resource that is being used.

→ A process needing the resource checks the semaphore to determine the resource's status & then decides how to proceed.

→ The most typical use of a semaphore is to protect a chunk of code that can only be executed one thread at a time.

Dijkstra's mechanism for mutual ~~exclusion~~ exclusion among an arbitrary no. of processes is called a semaphore.

A semaphore mechanism basically consists of two primitive operations

1) wait - V

2) Signal - P → S

Both operations operate on a semaphore variable 'S'. 'S' is initialised by the parent process then it is accessed and manipulated to P & V process.

Time	Process State			Mutex 1 = free 0 = busy	C.S! Attempt to enter
	P ₁	P ₂	P ₃		
M ₁	-	-	-	1	
M ₂	wait(s)	wait(s)	wait(s)	1	- : P ₁ , P ₂ , P ₃
M ₃	C.S	wait	wait	0	^{wait} P ₁ : P ₂ , P ₃ (C.S)
M ₄	signal(s)	wait	wait	1	- : P ₂ , P ₃
M ₅	remain- -der	C.S	wait	0	P ₂ , P ₃
M ₆	wait	signal(s)	wait	1	- : P ₃
M ₇	wait	remain- -der	C.S	0	P ₃

Parent Process

Var mutex: Semaphore (binary)

Process P₁

```

begin
while true do
begin
V wait (mutex);
Critical section
P signal (mutex);
end while
end; { P1 }

```

Process P₂

```

begin
while true do
begin
V wait (mutex);
Critical section
P signal (mutex);
remainder
end while
end; { P2 }

```

Process P₃

```

begin
while true do
begin
V wait (mutex);
Critical section
P signal (mutex);
remainder
end while
end; { P3 }

```


At time M_1 , before activity of 3 processes, parent process initialized semaphore variable, mutex variable to 1, indicating the availability of the shared resources to all.

At time M_2 , all processes are active and ready to enter their respective CS.

V function decrements

P function increments

All 3 waits start with testing the semaphore variable in an attempt to decrement it and proceed to their respective CS.

Semaphore rule does not satisfy which process will succeed.

Ex: Say P_1 succeed

P_1 immediately decrements the mutex variable & prevents other concurrent waits until it completes using the shared resource & CS (critical section).

Interprocess Communication

- 1) Interprocess synchronization - A set of protocols being used to preserve system integrity and consistently when concurrent process share resources.
- 2) Interprocess signalling - To exchange timing signals. It is used to exchange timing signals among concurrent process or threads, used to co-ordinate their collective progress.
- 3) Interprocess communication - Cooperating process must communicate for exchange of data, reporting program, accumulating results shared memory provides a simple and common means for interprocess communication.

11/02/21

synchronization:

②

Time	Process			Mutex 1: free 0: busy	CS
	P ₁	P ₂	P ₃		
A1	-	-	-	1	
M2	wait (s)	wait (s)	wait (s)	1	-: P ₁ , P ₂ , P ₃
M3	CS	wait	wait	0	P ₁ : P ₂ , P ₃ (CS)
M4	signal (s)	wait	wait	1	-: P ₂ , P ₃
M5	remained	CS	wait	0	P ₂ : P ₃
M6	wait	signal (s)	wait	1	: P ₃
M7	wait	remained	CS	0	P ₃

Parent Process

Var mutex: semaphore (binary)

Process P₁

```

begin
while true do
begin
wait(mutex)
critical section
signal(mutex)
end while
end {P1}

```

Process P₂

```

begin
while true do
begin
wait(mutex)
critical section
signal(mutex)
remainder
end while
end {P2}

```

Process P₃

```

begin
while true do
begin
wait(mutex)
critical section
signal(mutex)
remainder
end while
end {P3}

```


- ① In multi-tasking system, a semaphore is a variable with a value that indicates the status of a common resource.
- It is a lock as the resource that is being used.
 - A process needing the resource checks the semaphore to determine the resource status & then determines how to proceed.
 - The most typical use of semaphore is to protect a chunk of code that can only be executed by one thread at a time.

Program divides into sub parts → Fork
and o/p end → ~~wait~~ join

- Dijkstra's mechanism for mutual exclusion among ^{an} arbitrary no. of processes is called semaphore.

A semaphore mechanism ~~main~~ basically consist of 2 primitive operations:

1. wait $\rightarrow V$ ||
2. signal $\rightarrow P$ || $\rightarrow S$

Both the operations operate on a semaphore variable "s". "s" is initialized by the parent process then it is accessed & manipulated to P & V process.

③

At time M_1 , before activity of 3 processes parent process initialise the semaphore variable, mutex ~~variable~~ variable to 1. indicating the availability of shared resources to all.

At time M_2 , all processes are active and ready to enter their respective critical section. V function decrements & P function increments.

All 3 waits start with testing the semaphore variable in a attempt to decrement it and proceed to their critical section.

Semaphore ~~state~~ does not satisfy which process will succeed.

Ex: say P_1 succeed. P_1 immediately decrements the mutex variable & prevents other concurrent waits until it completes using the shared resource & critical section.

Inter process Communication:-

(4)

- 1) Interprocess Synchronisation → A set of protocols (rules) being used to preserve system integrity & consistency when concurrent process share resource.
- 2) Interprocess signalling → (To exchange timing signals)

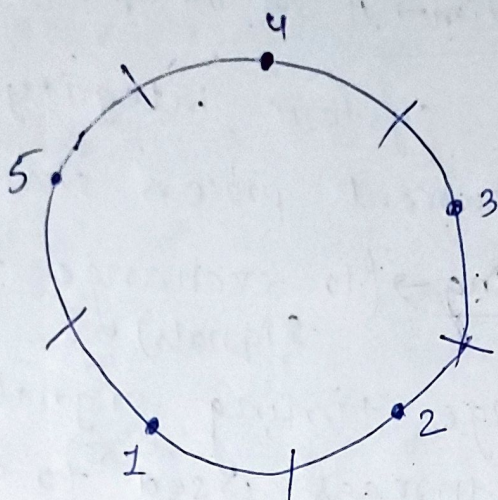
It is used to exchange timing signals among concurrent process or threads used to coordinate their collective progress.

Lamport's Global clock → sync. → ~~local~~ sync's local clocks of diff process

3) Interprocess Communication →

cooperating process must communicate for exchange of data, reporting progress, accumulating results shared memory provides a simple & common means for interprocess communication.

DINING PHILOSOPHER



1 & 3 $\rightarrow$ 4, 5 & 2 will starve & die

Think
 Hungry
 Eat

Wait for
 indefinite.

Starvation
 &
 Dead lock

starve
 &
 Die

1 & 3
1 & 4

$\rightarrow$ 2 & 5 starve & die.

1 & 4
3 & 5

condy
 soln

② will starve

My synchronization is 1, 2, 3, 4, 5

So, 1, 2, 3, 4, 5 should all be in alert state such that once 1 starts then after completion of 2 immediately starts and after completion of 2; 3 immediately starts and hence they are all synchronized and no one will starve & die.

Memory management :-

1. Keep track of resources
2. In MP who will get 1st
3. Allocate
4. Deallocate

1. Single Process monitor

2. Single contiguous allocation

3. Static Partition

4. Dynamic Partition

First fit
Best fit
Worst fit

space remains

Search partition specification table

proper space utilization

5. Relocation Algorithm

First fit can be

worst fit

Garbage collection & compaction

lot of unused space

6. ~~Paging~~ ~~Paging~~ Paging

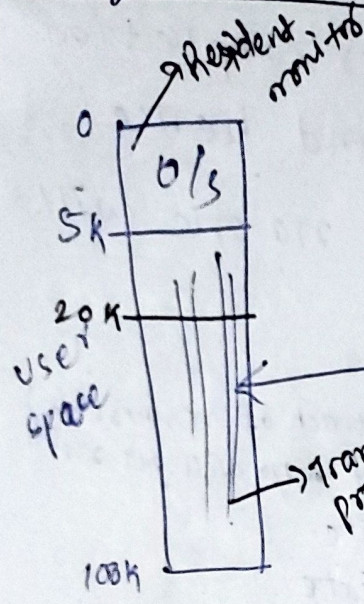
7. Segmentation

8. Segmentation with paging

virtual memory allocation

Single Process Monitor

ROM
RAM
EPROM
EEPROM



Either o/s can be stored in upper part or lower part.

only single program will be allocated here

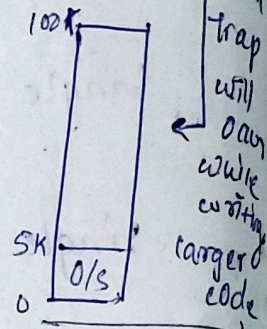
Not more than 1 can be allocated.

memory

$$100 - 5 = 95K$$

$$J_1 \rightarrow 20K$$

75K $\rightarrow$ unnecessarily wastage of memory



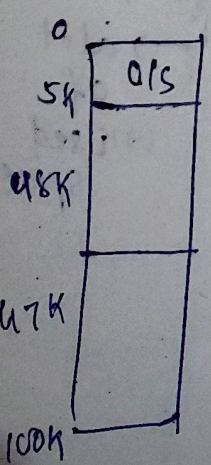
Simple b/w

Easier to implement

2 ways to protect
 $\rightarrow$ Fence req.
 $\rightarrow$ Reference bit (0 or 1)

Single contiguous allocation

$\rightarrow$ divides memory into 2 parts of the user mode.



$\rightarrow$ other part goes waste.

$\rightarrow$ Program of 49K cannot be allocated in any part

Dividing to support multiprogramming but program can be allocated only in one part and other part remains ~~remains~~ Unutilized.

only one process can be allocated at a time.

Synchronization :-

There are various technique to have process synchronization:

- ① TEST AND SET INSTRUCTION
- ② WAIT AND SIGNAL MECHANISM
- ③ P AND V SEMAPHORE
- ④ CRITICAL REGION

1. TEST AND SET INSTRUCTION:

In this mechanism there used a lock byte (or) semaphore. For each shared data base (or) device there should be a separate lock byte. If the lock byte = 0 means the resource is available, where as lock byte = 1 means the reference is already in use.

To operate on such shared resource a process must perform these function

- STEP:
- ① Examine the value of the lock byte i.e. 0 or 1
 - ② If it is '0' set the lock byte to 1
 - ③ If it is '1' go back to step ①

After the process has completed its use of the resource, it sets the lock byte to zero.

② WAIT AND SIGNAL MECHANISM:

The wait and signal mechanism is taken because of the disadvantage of Test & SET instruction i.e. unnecessarily a loop is there to check the lock byte and

waiting to change to zero.

In case of wait and signal mechanism there are two conditions lock and unlock.

LOCK(X)

1. Examine the value of the lockbyte i.e. either it is '0' or '1'
2. Set the lock byte to '1'
3. If the original value is '1', call wait(X).

UNLOCK(X)

1. Set the lock byte to '0'
2. Call signal(X).

A wait(X) sets the process PCB to blocked (wait) state and links it to the lock byte x. Another process is then selected to run by the process scheduler.

A signal(X) checks the blocked list associated with lock byte 'x'. If there are any process blocked waiting for 'x', one is selected and its PCB is set to the ready state so that the process scheduler will select this new process for execution.

③ P AND V SEMAPHORES:

Semaphore is a special purpose datatype introduced by Ediger Dijkstra (1965).

The letter P & V derive from the Dutch words used in the original description.

A semaphore is a non-negative integer on which only initialisation and the indivisible operations P and V are defined.

There are two operations on a semaphore:

1. Wait (or) P-operation (or) down operation
2. Signal (or) V-operation (or) up operation

The V operation increases the value by one and in general indicates that a resource has become busy. The V operation decreases the value by one when that can be done without the value getting negative and in general indicates resource is free.

A semaphore operation has the effect of either:

- ~~1. Incr~~ 1. Incrementing/decrementing the value of a semaphore by 1.
- (or) 2. Blocking (waiting)/~~awakening~~ a process

Process blocking is implemented by interfacing with the process scheduling component of the OS. Thus a process is suspended which was executing on the CPU.

This avoids the busy wait problem.

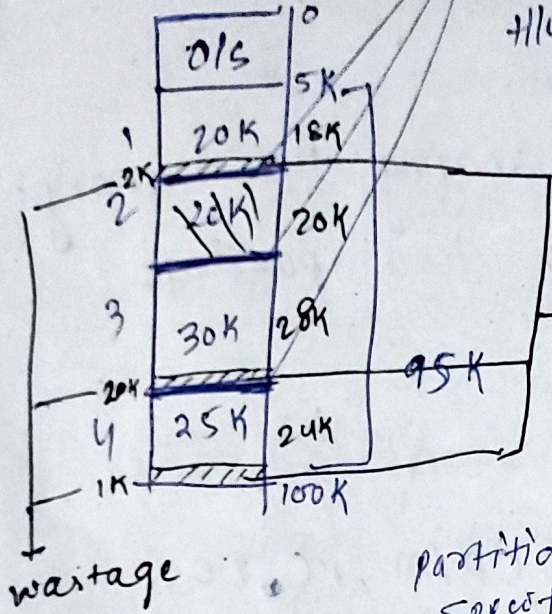
The semantics of the 'P' and 'V' operation executed on a semaphore 's' are as follows:

P operation: $P(s)$ - If $s > 0$ then $s := s - 1$
else block the process
executing the 'P' operation

V operation: $V(s)$ - If there exist processes
blocked on 's' then wake ~~one~~
one process
else $s := s + 1$

2402

fence register $\rightarrow$ Blw any 2 jobs
 Memory $\rightarrow$ 4 parts
 H/w



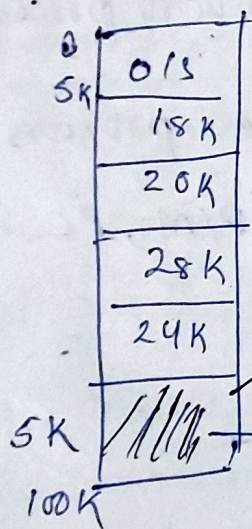
Divide memory $\rightarrow$ able to support multiprogramming

Internal Fragmentation $\rightarrow$ Static Partition

$\hookrightarrow$ Unused space blw any 2 jobs.

partition specification table

$\hookrightarrow$ info. about partitions



Dynamic Partition

$\hookrightarrow$ Jobs stored one by one contigiously.

External fragmentation

$\hookrightarrow$ if any job does not arrive of $\leq 5K$

Base + Offset $\rightarrow$ Address space

MEMORY MANAGEMENT

Single Process Monitor -

It is a simplest way for managing the memory.
Here memory is divided into two portion:

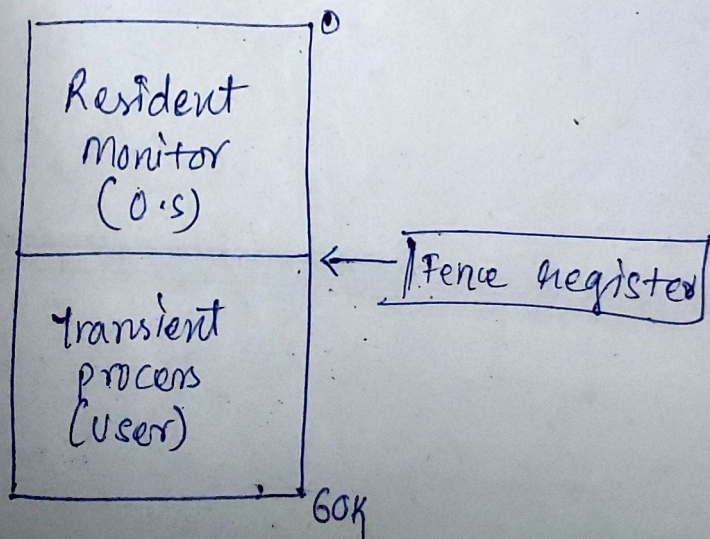
- (i) Resident monitor
- (ii) Transient process

Gary Kildall (1974) created CP/M, PC DOS
~~(Computer Programming for microcomputer)~~
Control program for micro computer/micro processor

This process is generated by a single process
microcomputer operating system i.e. CP/M, PC DOS
during:

- (a) Abnormal execution
- (b) Completion of work

Here trap vector table is used to design the
memory i.e. O.S is present upper (or) lower part.



Advantages: ① Simple

② Easy to implement

③ No special hardware is required for contiguous allocation

Disadvantages: ① User's process is limited to available space.

② Poor utilization of memory space

③ Can't support multiprogramming

④ Poor utilisation of processor

How to Protect Hardware:

If the monitor is residing in low memory and a user program executing in high memory, we need to protect the monitor code and data from changes (accidental or malicious) by the user program.

There are two ways to protect it

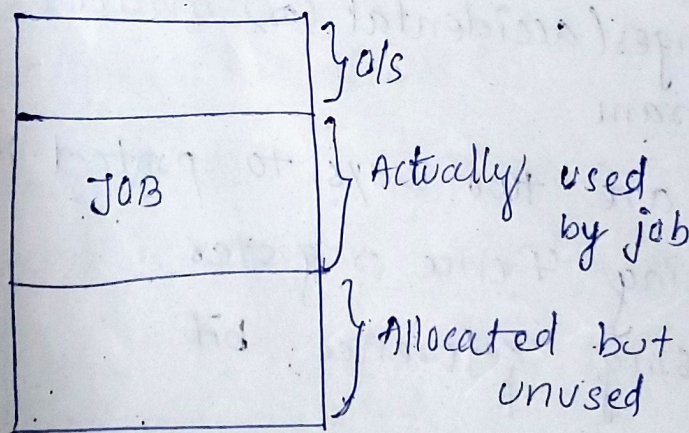
① Using Fence register

② Using Reference bit

Single Contiguous Allocation :

Single contiguous allocation is usually associated with minicomputers with simple batch operating systems. In such system there is no multiprogramming and a one-to-one correspondence exists between a user, a job and a process.

Here memory is divided into three contiguous regions. A portion of memory is permanently allocated to the operating system. The job actually uses only a portion of the allocated memory, leaving an allocated but unused region of memory.



A single contiguous allocation

Function : ① Keeping track of memory i.e. it is allocated entirely to one job.

② Determining factor on memory policy i.e. the job will get all memory when scheduled.

- ③ Allocation of memory i.e. all of it is allocated to the job.
- ④ Deallocation of memory i.e. when the job is done, all memory is returned to free status.

Advantages: ① It is very easy to understand

② It is simple.

Disadvantages: ① Poor utilization of processes

② Poor utilization of memory

③ The user's job is limited to the size of available main memory.

PARTITION ALLOCATION:

Partition allocation is one of the simplest memory management techniques for supporting multiprogramming. Here main memory is divided into separate memory partitions. Each partition holds a separate job's address space.

By using bound register (fence register) we can protect one job from either the O/S (or) other job. So if the job tried to access memory outside the partition, a portion interrupt will occur.

Function:

- ① keeping track of the status of each portion
- ② Determine who will get memory - this is handled largely by the job scheduler.
- ③ Allocation - an available portion of sufficient size is assigned.
- ④ Deallocation - when the job technique terminates, the partitions is indicated "not in use" and is available for future allocation.

Disadvantage:

- ① The bound registers must be changed every time that the processor is reassigned for multiprogramming.
- ② It is difficult to extend protection to the I/O channel.

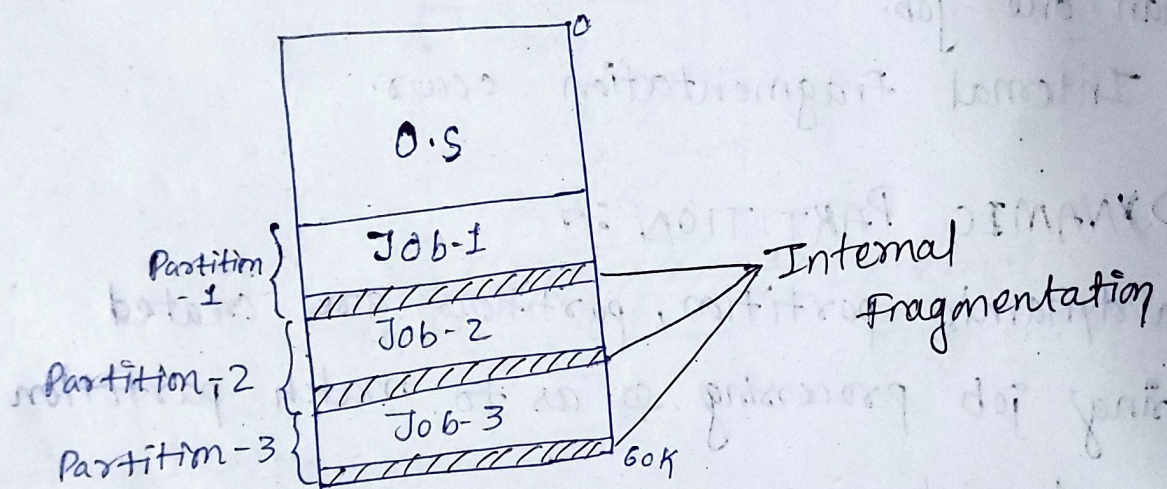
PARTITION SPECIFICATION:

The partition are of two types:

- ① Static Partition
- ② Dynamic Partition

STATIC PARTITION:

Here the memory is divided into partitions prior to the processing of any job. Here there is a partition specification table, which specifies the size, location and status of job. The partition sizes are chosen such that there are closely to the most common job size.



Advantage:

- ① It facilitates multiprogramming, hence more efficient utilization of the processes and I/O devices.
- ② It requires no special costly hardware.
- ③ The algorithms used are simple and easy to implement.

Disadvantages:

- ① Inefficient use of memory space since there is no correspondance between the partition size and the incoming job size.
- ② It require more memory than a single contiguous allocation system (as well as a more complex operating system) in order to hold more than one job.
- ③ Internal fragmentation occur.

DYNAMIC PARTITIONING

In dynamic partition, partitions are created during job processing so as to match partition sizes to job sizes.

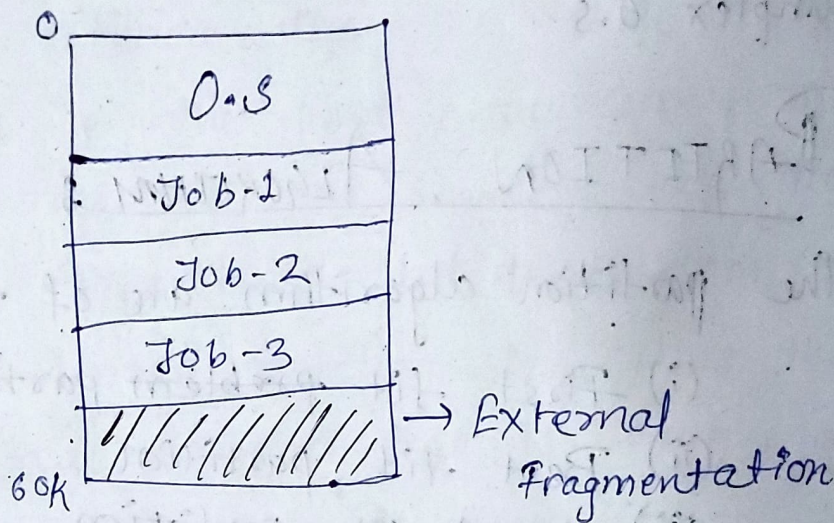
Here job can be selected by multiprogramming. Mainly in dynamic allocation, external fragmentation will create.

When we have to allocate a program, we must follow the following steps:

- ① A free area at least as large as the partition desired must be found.
- ② If the area is larger ~~the~~ than the area that needed, then it must be spilt into two pieces.

- (a) becomes the allocated partition
- (b) becomes a smaller free area.

But when a partition is deallocated we try to merge it with any adjacent free areas so as to make one contiguous free area rather than many small pieces.



Advantages:

- ① It requires no special costly hardware.
- ② It facilitates multiprogramming hence more efficient utilization of the processor and I/O devices.
- ③ The algorithms used are simple and easy to implement.
- ④ Small amount of memory is wasted.

Disadvantages:

- ① It is time taking to watch partition sizes to

job sizes.

② External fragmentation can be a significant problem.

③ It requires more memory than a single contiguous allocation system as well as more complex OS.

complex O.S.

PARTITION ALGORITHM:

The partition algorithm are of three type

- (i) First fit ~~problem~~ partition
- (ii) Best fit partition
- (iii) Worst fit partition

First Fit Partition:

In first fit partition the free table is kept sorted by location i.e. each consecutive entry correspond to a free area that ~~a~~ partition starts at a higher memory address. When it is necessary to allocate a partition, we start at the free area at the lowest memory address and keep looking until we find the first free area big enough for the

partition - to fit.

Advantages:

- ① Execution speed is faster
- ② Time complexities is reduced

BEST FIT PARTITION:

In best fit partition, the free table is kept sorted by size i.e. the first entry correspond to the free area with the smallest size. Here the free area closet inside but larger than (or) equal to the desired partition size. Thus the first free area we find that is large enough for the desired partition is best fit. In best fit it search all the entry of the PDT (Partition Definition Table).

Advantages:

- ① On the average, the best fit free area can be found by searching only half the table
- ② If ~~the~~ there is a free area of exactly the desired size, it will be selected.
- ③ If there is no free area, the partition is carved out of the smallest possible free area & does not destroy a large free area.

Disadvantages:

- ① the resulting free area is ^{often} quite small. So it is worthless.

WORST FIT PARTITION:

It allocate the largest hole. Again we must search the entire list. Unless it is sorted by size. This strategy produce the largest left-over hole, which may be more useful than the smaller left-over hole from a best fit approach.

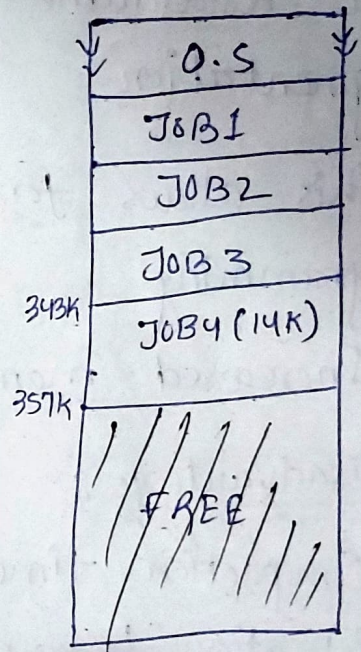
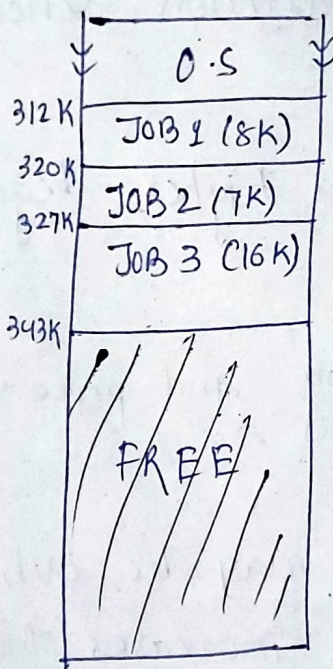
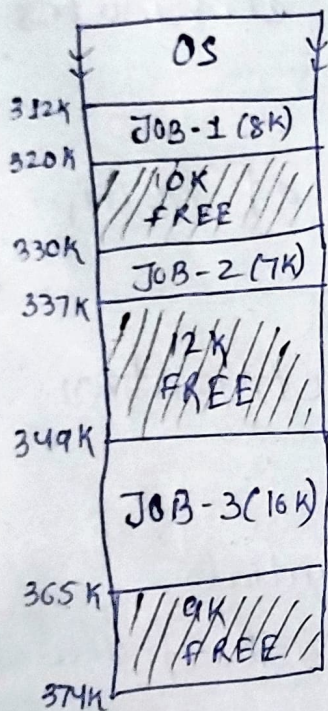
This algorithm suffer from external fragmentation. As files are allocated and deleted, the free disk space is broken into little pieces. External fragmentation exists when enough total disk space exists to satisfy a request but it is not contiguous; storage is fragmented into a large no. of small holes.

REALLOCATABLE PARTITION

Base + Offset

→ Address space

An obvious solution to the fragmentation problem is to periodically combine all free areas into one contiguous area. This can be done by moving the contents of all allocated partitions so that they become contiguous. As shown below.



(After compaction)

Moving a job's partition, doesn't guarantee that job will still run correctly because there are many location-sensitive items such as

① Base register ② memory referencing ③ Parameter lists

④ Data structure

that used address pointers. To operate correctly, all location-sensitive items must be suitably modified.

This process of adjusting location-sensitive address is called relocation. Except for the process of compaction & relocation partition, allocation schemes are very similar to the ~~part~~-partitioned allocation techniques.

Advantages:

- ① The relocatable partition scheme eliminates fragmentation.
- ② This allows for a higher degree of multiprogramming.
- ③ Increased memory and processor utilization.

Disadvantage:

- ① Compaction time may be substantial.
- ② Relocation hardware increases the cost of the computer.
- ③ A job's partition size is limited to the size of physical memory.

NB: Hardware of this type available on the Burroughs 5500 & 6700

Also known as 'garbage collection & compaction'

26/02

Real Time OS :-

Given task will be completed (in specified time) with in ~~stipulated~~ ^{stipulated} Deadline.

otherwise it ~~will be~~ system will be a ^{run down and it would} catastrophic failure.

Real time $\rightarrow$ Dynamically faster execution.

Kernel $\rightarrow$ core part of OS.

It is a interface b/w user program & h/w.

Also take care that there should not be any conflict b/w processes and b/w processors.

$\rightarrow$ Distributed OS?

$\rightarrow$ N/w OS?

$\rightarrow$ Interactive OS?

OS will be generated that should be within a ~~st~~ stipulated deadline.

Two types of Real Time OS

① Hard RTOS

② Soft RTOS

$\rightarrow$ Rocket $\rightarrow$ O/P.

$\rightarrow$ Time taken to load some webpage.

Distributed OS

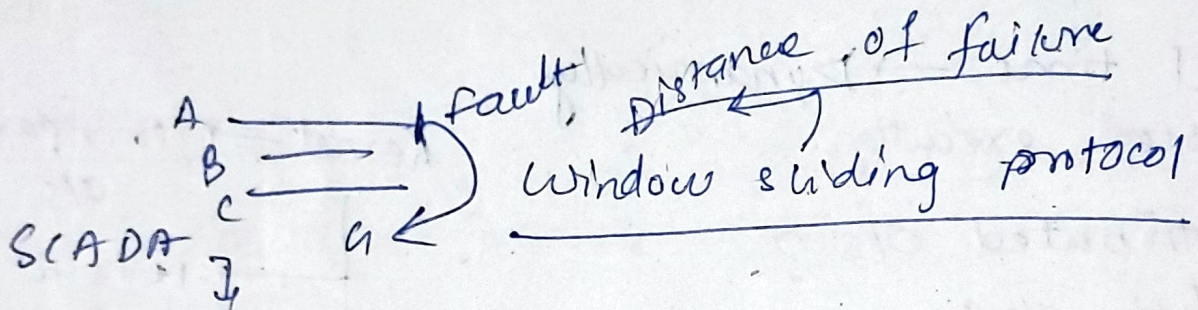
$\rightarrow$ OS is scattered in diff. servers i.e. distributed in various geographical regions etc and if one server is lost it hardly matters.

Interactive OS $\rightarrow$ one thread jumps to other App. and other jumps to another application.

N/w o/s $\rightarrow$ Graphic design uses n/w o/s

$\hookrightarrow$ Within a geographical region various system works in master slave concept.

master $\swarrow$ $\searrow$ Dumb terminal.

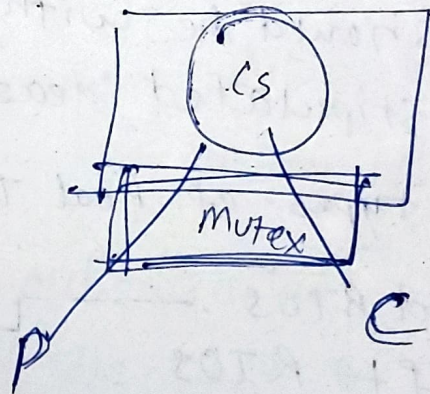


Supervisory control and data execution

PLC $\rightarrow$ Program ^{enable} Logic control

$\Rightarrow$ Producer consumer

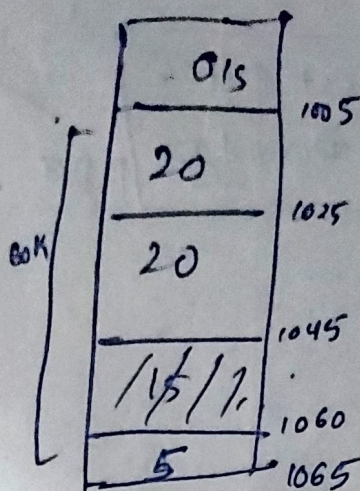
CS $\rightarrow$ shared resource



CPU burst time
& I/O burst

PDT

Partition name	Size	Addr.
1	20	
2	20	
3	15	
4	5	



Job $\rightarrow$ 15K $\rightarrow$ arrives

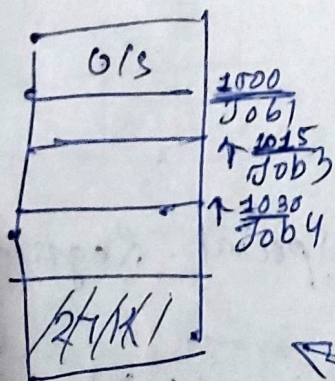
Check PDT (if 15K is available)

Allocate $\rightarrow$ 15K. $\rightarrow$ 1045 $\rightarrow$ 1060.

if put in 20K $\rightarrow$ 1 $\rightarrow$ 5K will be lost without searching.

17K $\rightarrow$ Job $\rightarrow$ arrives. $\rightarrow$ to

$\rightarrow$ Job arrives and without checking PDT.
First fit

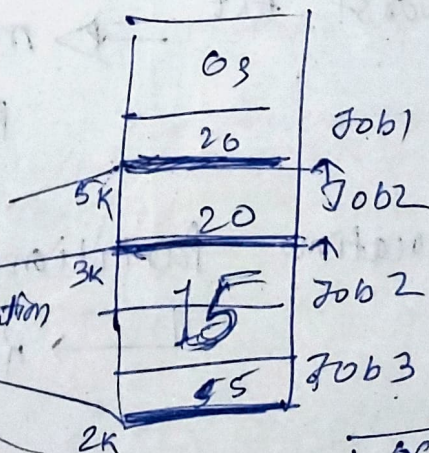


mp
&
Confusion
State

01s will search in current address.

Job 2 releases

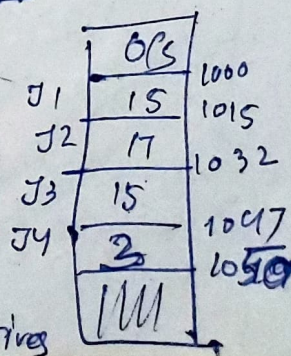
Frag-
mentation



10417 = 27K

J2 releases

2 20 3 4
J5, J6, J7, J8 arrives

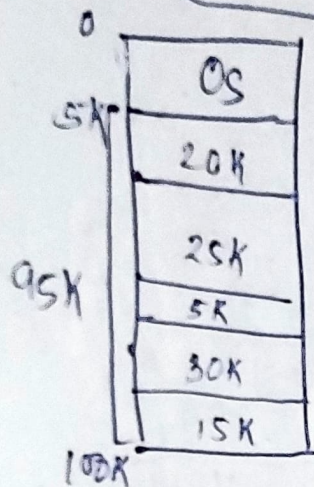


12/03

First Fit, Best Fit & Worst Fit

It will also consume some memory

PDT → Partition division table



Job	
1	20
2	25
3	05
4	30
5	15

1st Fit → If 10K job arrives then the 1st find which is sufficient enough to allocate even if there is wastage of memory

Best Fit → Min^m wastage after going through PDT

Worst Fit → Max^m wastage after PDT

Reallocation

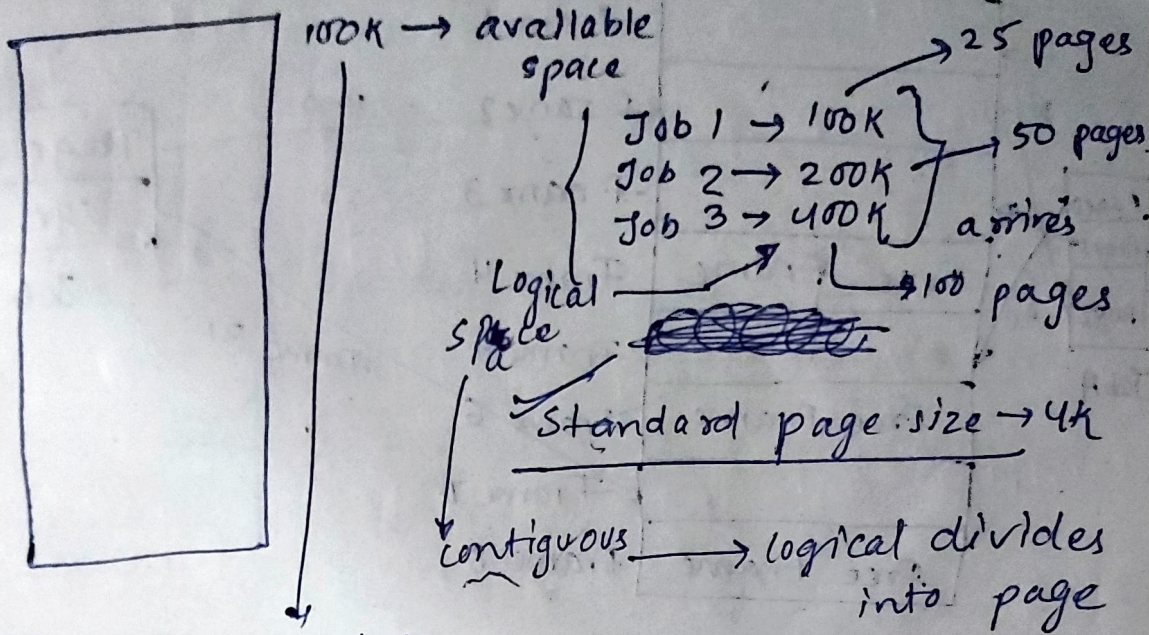
Partition:

→ Register, Special Registers

↓
used to identify the new address after reallocation.

Virtual memory concept

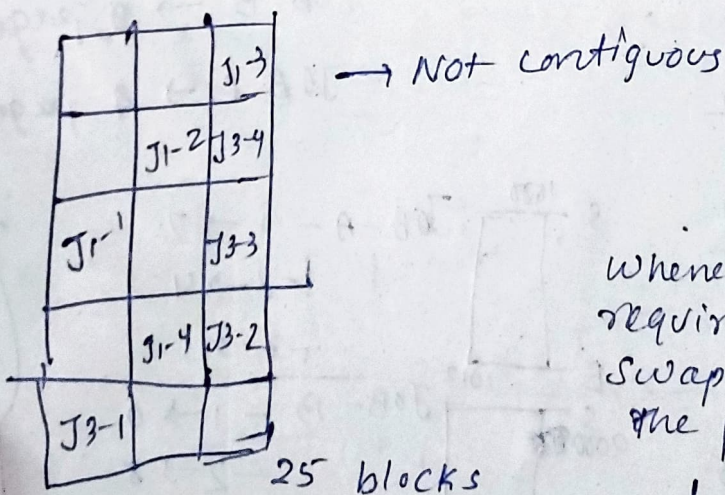
Paging $\Rightarrow$ $\rightarrow$ Divide equal no. of page.



Physical memory Divided into 25 ~~pages~~ block

may not be contiguous

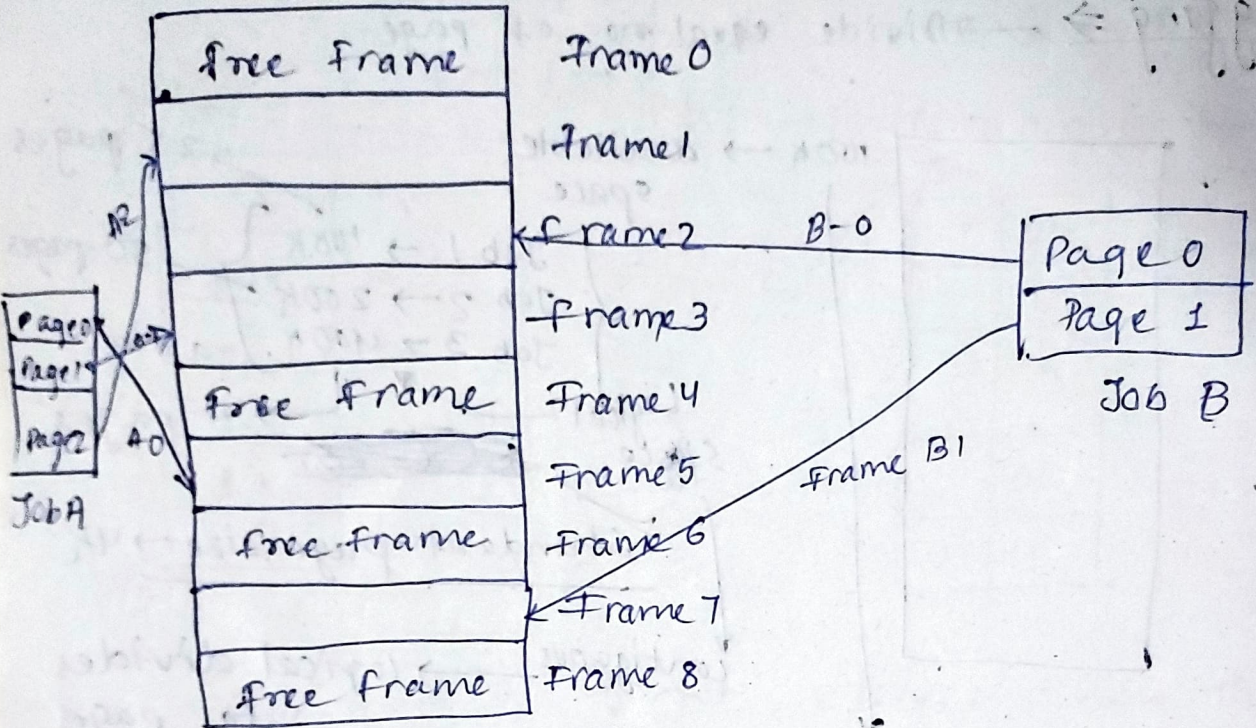
It is not required to bring whole program at a time but some part of it



Pages allocated in the block

Swap in
swap out
Whenever required swap in the pages
after completion
& other can be swapped in

memory space



19/03

9 Frames

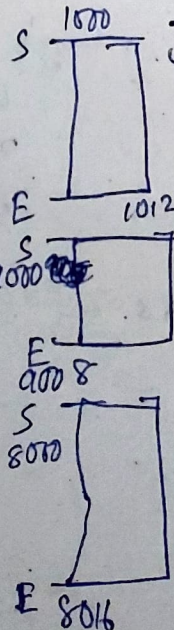
logically contiguous but physically not

JOB-B-1	0
JOB-A-3	1
JOB-C-4	2
JOB-C-1	3
JOB-A-2	4
JOB-C-3	5
JOB-C-2	6
JOB-A-1	7
JOB-B-2	8

JOB A → 3 pages

JOB B → 2 pages

JOB C → 4 pages



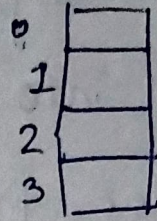
JOB-A-1 → 7
 -2 → 4
 -3 → 1
 JOB-B-1 → 0
 -2 → 8
 JOB-C-1 → 3
 -2 → 6
 -3 → 5
 -4 → 2

Another job is arriving $\rightarrow$ JOB-D $\rightarrow$ 14K

$\therefore$ 4 pages

To allocate this other jobs must be swapped out

$\downarrow$
This is page
replace algo.



- ① ~~FIFS~~ $\rightarrow$ FIFO $\rightarrow$ First ~~come~~ ⁱⁿ first out
- ② LRU $\rightarrow$ Least recently used.
- ③ LFU $\rightarrow$ Least frequently used

(Not used recently)

$\downarrow$
(Not needed pages)

Page hit

Page Fault

DMA $\rightarrow$ Direct memory access

$\downarrow$
cycle stealing
 $\rightarrow$ Directly accesses pages

if Job-C $\rightarrow$ 17K $\rightarrow$ 5 pages

1 $\rightarrow$ 3
2 $\rightarrow$ 6
3 $\rightarrow$ 5
4 $\rightarrow$ 2
5 $\rightarrow$

$\rightarrow$ all cannot be loaded at a time

$\rightarrow$ page hit

$\rightarrow$ not get

$\downarrow$
Page miss

page ~~not~~ should be swapped out from memory to allocate this

Paging :-

In page memory management each ~~page~~^{address} ~~job's~~^{page} is divided into equal pieces called pages.

and likewise physical memory is divided into pieces of same size called block (or frames)

Then by providing a suitable h/w mapping facility any pages can be placed into any block.

Page Displacement diagram)

(Page 2 displacement 24)

The pages remain logically contiguous (i.e. to the user program) but the corresponding blocks are not necessarily contiguous.

As in the relocatable partition scheme the user programs are ~~not~~ unaffected by mapping because it has no visible effect upon the address space.

for the h/w to perform the mapping from address space to physical memory there must be

a separate register for each page, the registers are often called page maps or page mapped tables (PMT).

They may be either special h/w registers or a reserve section of memory.

Functions:

→ Keep track of the status - Accomplished through two sets of table

① Page mapped Table
② memory block table (MBT)

Starting & Ending addr. Job, Block no.

→ Determining who will get memory.

→ Allocation - All the pages of the job must be loaded in the assigned block and appropriate entries made in the ~~PMT~~ PMT & MBT

→ Deallocation - when the job is done the blocks must be returned to free status by ~~entering~~ adjusting the entries in the Block Table.

Adv.

→ The page memory ~~eliminate~~ ^{scheme} eliminates ^{major} fragmentation.

→ This is allowed for high degree of multiprogramming.

→ Increase the memory & processor utilization.
(Due to this remaining busy in swap in & swap out & will not get time to process
This is known as Thrashing)

→ The compaction time is also eliminated

Disadv :

- The page address mapping b/w usually increase cost of computer
- memory must be used to store various tables
- Process time must be expanded to maintain and update these tables.
- Job's address space is limited to size of the physical memory.
- ^{still} We get the internal fragmentation.

Memory management schemes

① Simple Paged allocation

② Demand Paging

③ Segmentation

④ Paged segmentation.

→ Logical address is, the address generated by the CPU and the range of these address is called Address space / user's Address space.

Where as physical address is the actual address of the process in the main memory and the range of these addresses is called Memory space / Physical Address space.

Simple Paged Allocation -

- All pages for a job should reside in memory.
- Divides the job address space into pages of the same size.
- Divides the main memory address space into blocks / frames
- Pages are units of memory that are swapped in and swapped out of primary memory.
- Pages are grouped together for a given user

job and are placed in page tables.

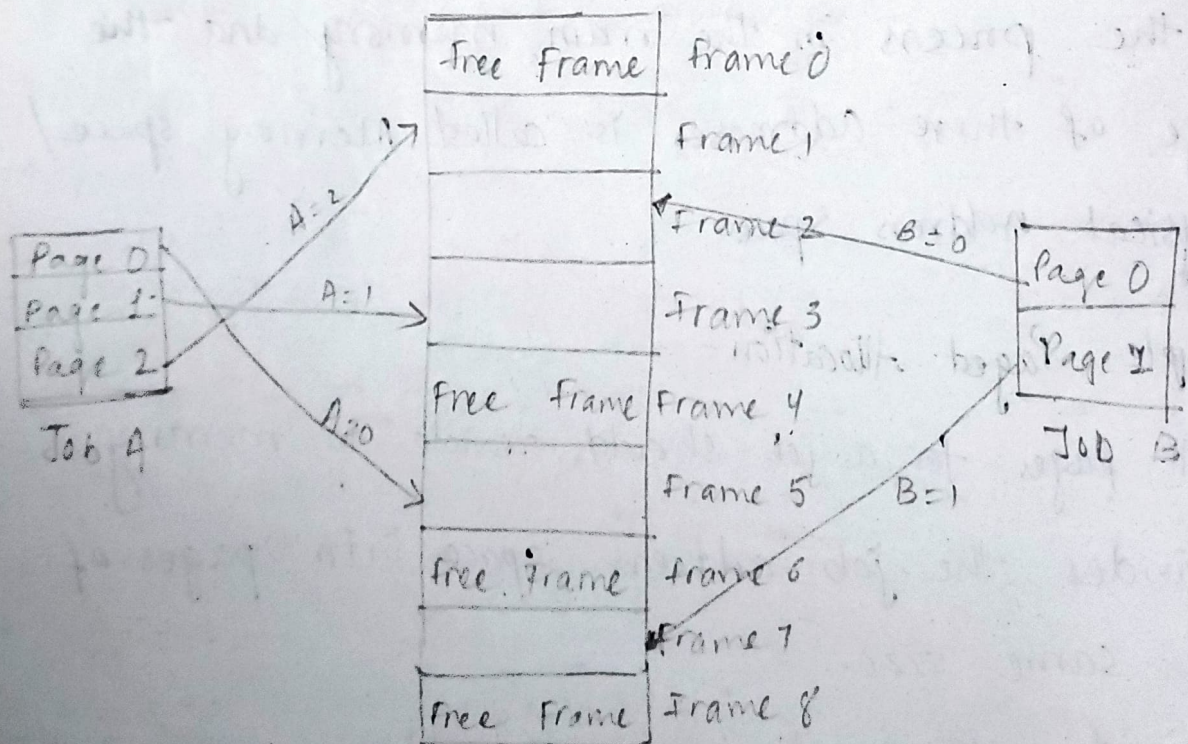
Adv: ① Simple paged allocation is a solution to fragmentation.

② As each page is separately allocated, the users job need not be contiguous.

Disadv: ① Extra ~~memory~~ ^{memory} required for storing page table.

② Considerable amount of h/w support is required for address transformation etc.

③ All pages of entire job must be in memory.

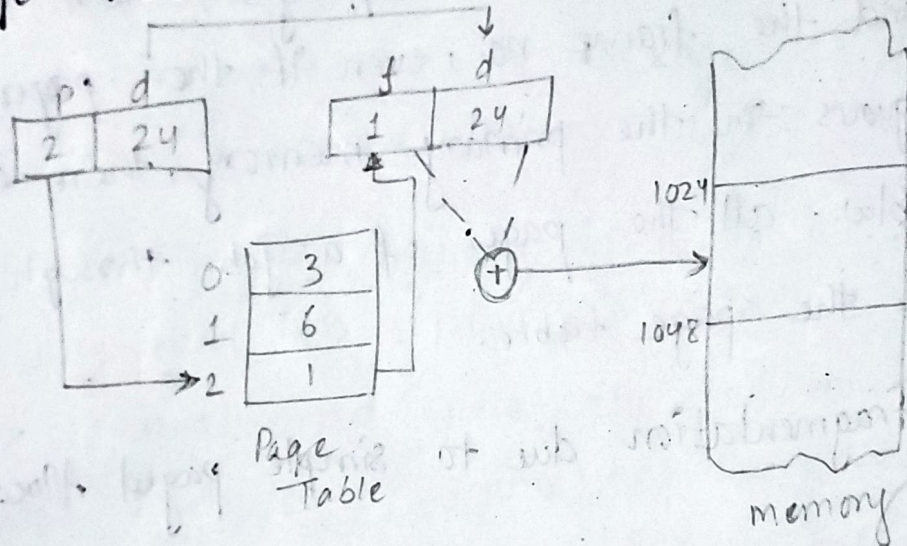


The example above ~~the~~ ^{memory space} shows a page map table for 2 jobs viz. page no. and frame no. which essentially shows the mapping b/w page no. & block no. Job A has 3 pages viz- 0, 1, 2. Page 0 maps to frame 5 in the OS.

page 1 to frame 3 and page 2 to frame 1 in the OS:
 JOB. B has 2 pages viz. page 0 in frame 2 & page 1 in frame 1.

Thus, we can observe that pages of a job need not be located contiguously in the memory.

Page Table and Address Translation:

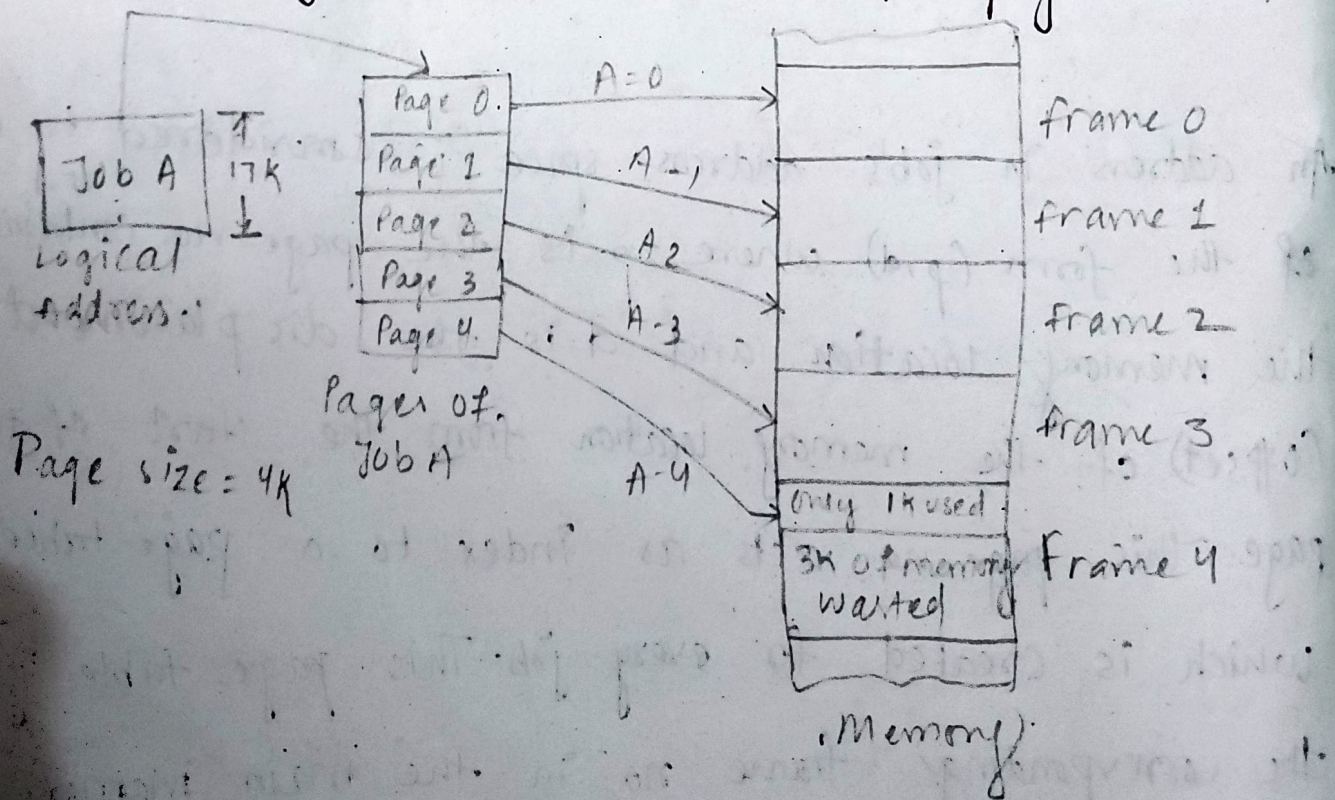


An address in job's address space is considered to be of the form (p, d) where p is the page no. containing the memory location and d is the displacement (offset) of the memory location from the start of the page. This page no. acts as index to a page table which is created for every job. This page table gives the corresponding frame no. in the main memory for the particular page. This frame no. with the displacement would give the physical address.

The page table has two purposes:
 It has got entries which map each page no. in the program (in the secondary memory) and the corresponding page frame no. (or block no. present in the primary memory).

Since there is one to one mapping between the page no. and the frame no., even if the pages are not contiguous in the primary memory, there exists a link b/w all the pages of a job through this mapping in the page table.

Internal fragmentation due to simple paged Allocation:



Fragmentation is wastage of memory, it occurs when memory is divided into many small chunks/free blocks and these free blocks are not able to satisfy any request.

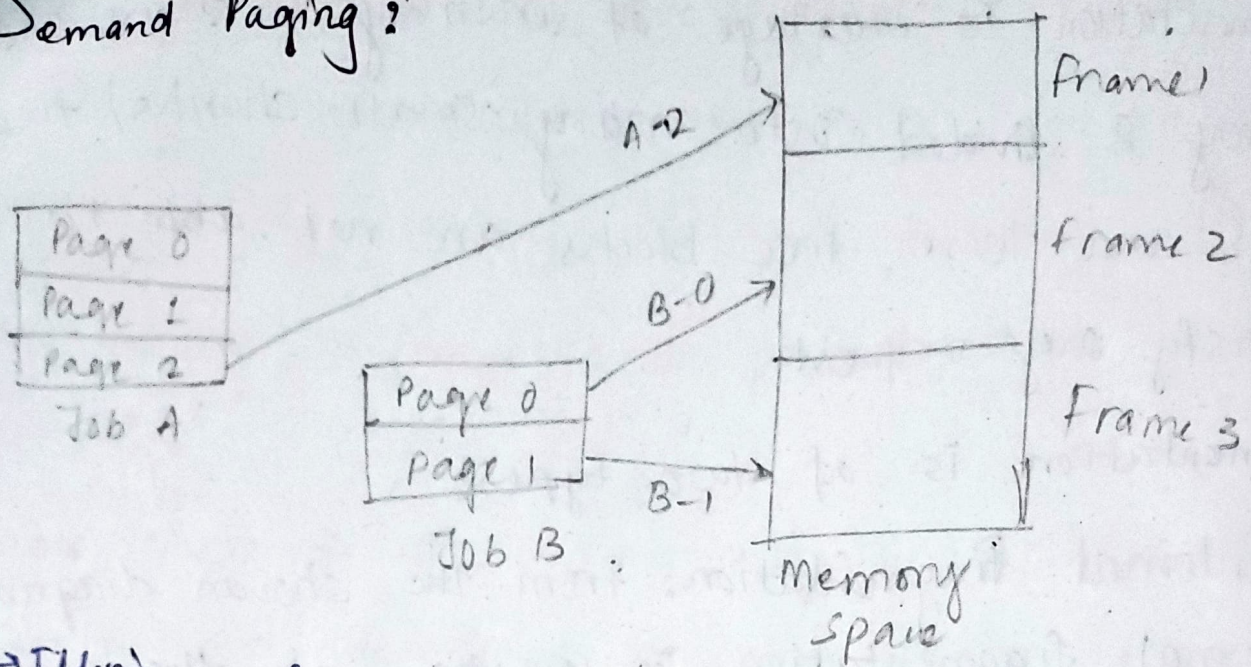
Fragmentation is of two types:

① Internal Fragmentation: From the shown diagram, internal fragmentation is wastage of the space inside the allocated memory blocks.

Because page size/memory block is of fixed size. Job may need less space but min^m 1 memory block is allocated (space allocated to a job is always multiple of frame), so the wastage of memory happens is Internal fragmentation. Limitation because of fixed size memory block.

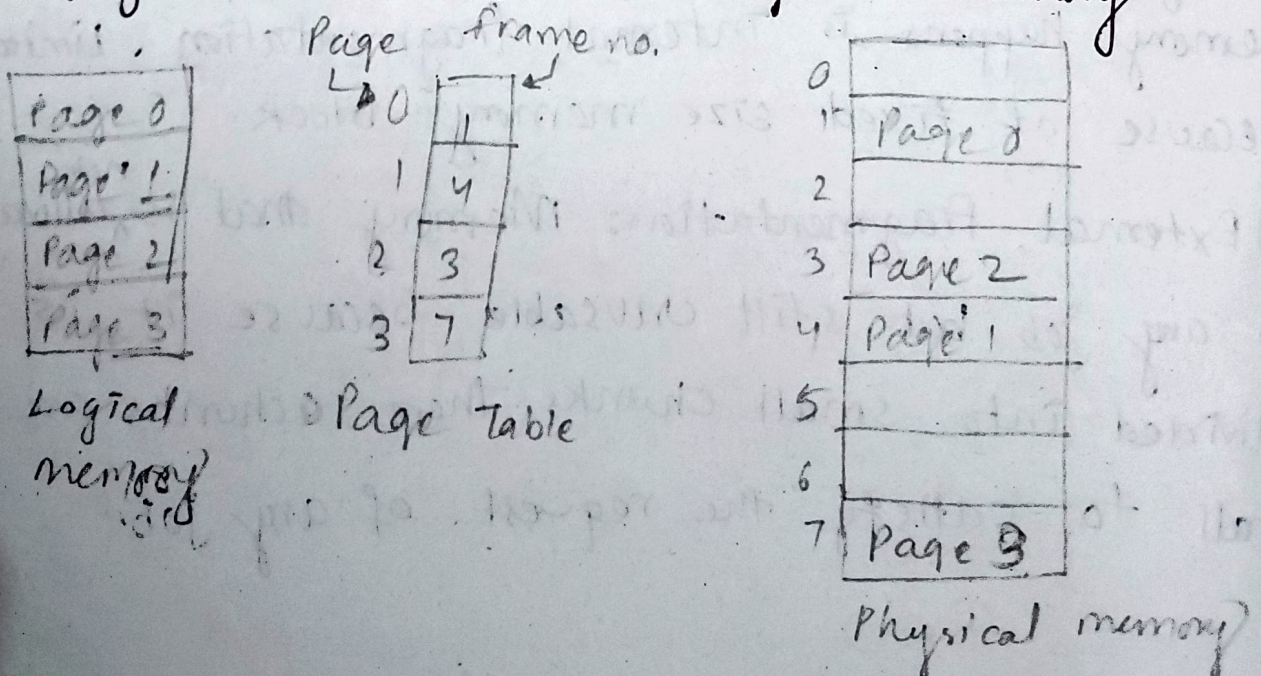
② External Fragmentation: Memory that is ^{not} allocated to any job but still unusable because it is divided into small chunks. These chunks are too small to satisfy the request of any job.

Demand Paging:



- Illusion of infinite memory available to the user (virtual memory) which
- Job operating under demand paging
- A page is brought to the main memory only when there is a demand.

Paging model of logical & Physical memory



Logical memory

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p

Page no

frame no

0	5
1	6
2	1
3	2

Physical memory

0	
4	a b c d
8	e f g h
12	
16	
20	a b c d
24	e f g h
28	

∴ Logical memory $\rightarrow (0-15)K$

Page $\rightarrow (0 \rightarrow 3)$

Frame (given)

Page $\rightarrow 0 \rightarrow$ Frame $\rightarrow 5 \Rightarrow 5 \times 4 = 20 \rightarrow 0 \rightarrow a, b, c, d$

1 $\rightarrow$ Frame $\rightarrow 6 \Rightarrow 6 \times 4 = 24 \rightarrow 1 \rightarrow e, f, g, h$

2 $\rightarrow$ Frame $\rightarrow 1 \Rightarrow 1 \times 4 = 4 \rightarrow 2 \rightarrow i, j, k, l$

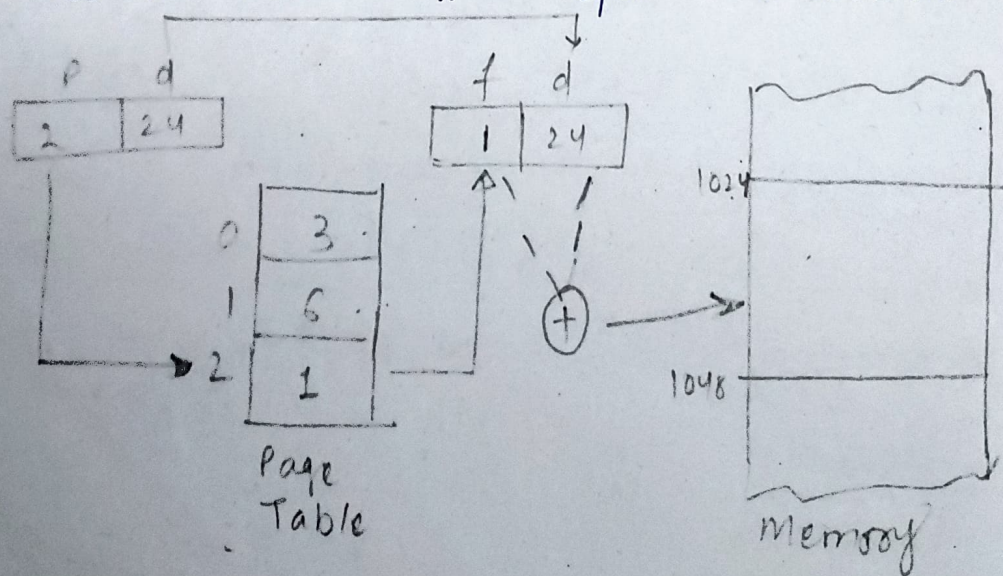
3 $\rightarrow$ Frame $\rightarrow 2 \Rightarrow 2 \times 4 = 8 \rightarrow 3 \rightarrow m, n, o, p$

↓
Physical memory

Paged Memory Management :

In paged memory management each job's address space is divided into equal pieces called pages and likewise physical memory is divided into pieces of same size called blocks. Then by providing a suitable h/w mapping facility, ~~any~~ any page can be placed into any block. The pages remain logically contiguous (i.e. to the user program) but the corresponding blocks are not necessarily contiguous.

As in the relocatable partitioned scheme, the user programs are unaffected by the mapping because it has no visible effect upon the address space.



For the h/w to perform the mapping from address space to physical memory, there must be a separate register for each page; These registers are often called

page maps or page map tables (PMTs). They may be either special h/w. registered or a reserved section of memory.

FUNCTIONS:

- ① Keeping track of status - Accomplished through two sets of table:
 - Ⓐ Page map table (PMT)
 - Ⓑ memory block table (MBT)
- ② Determining who gets memory.
- ③ Allocation - All pages of the job must be loaded into assigned blocks and appropriate entries made in the PMT & MBT.
- ④ Deallocation - When the job is done (completed), blocks must be returned to free status by adjusting entries in block table.

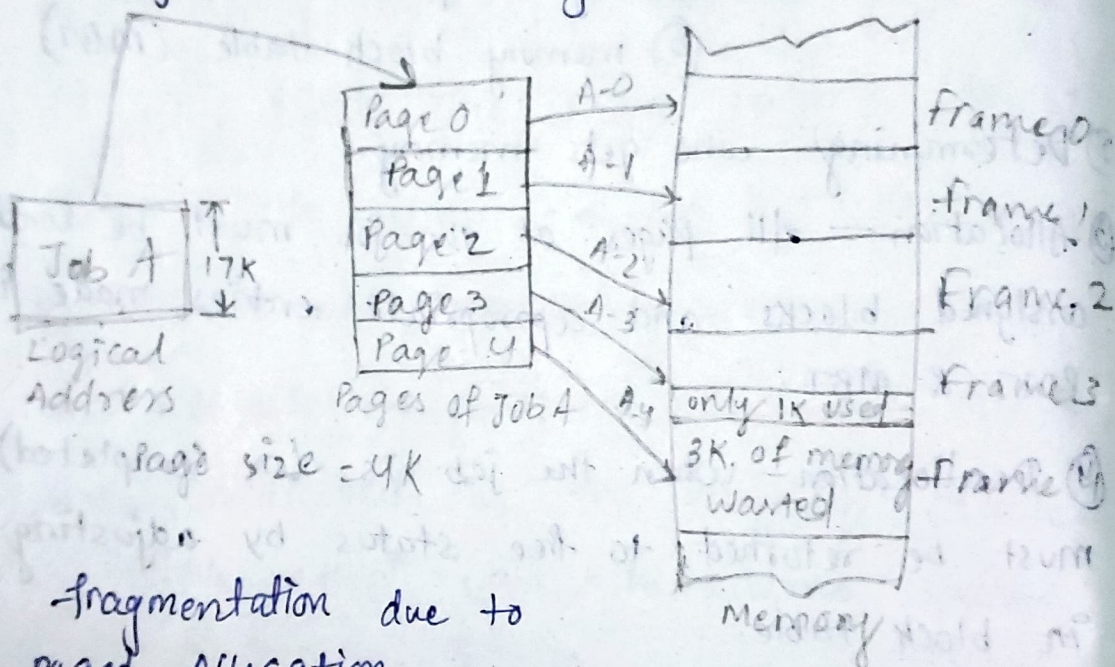
Advantages:

- ① The paged memory scheme eliminates major fragmentation.
- ② This allowed for high degree of multiprogramming.
- ③ Increased the memory and processor utilization.
- ④ The compaction time is also eliminated.

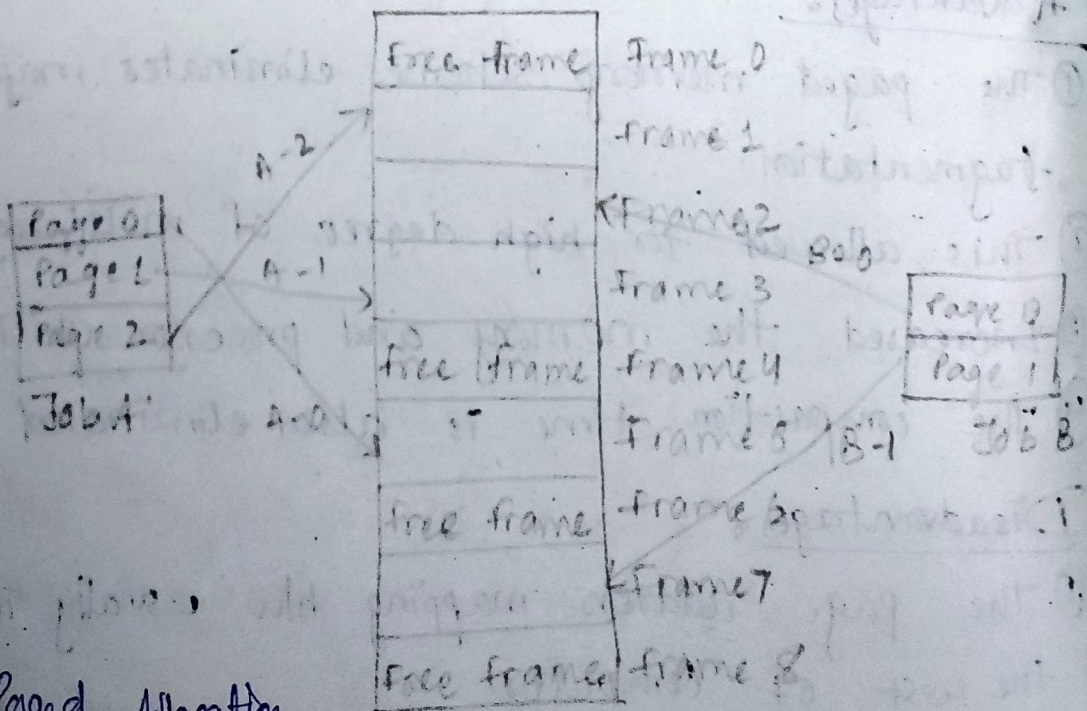
Disadvantages:

- ① The page address mapping h/w usually increase the cost of computer.

- ② Memory must be used to store the various tables.
Processor time must be expended to maintain and update these tables.
- ③ Job's address space is limited to the size of physical memory.
- ④ Still we get internal fragmentation.



Internal fragmentation due to simple paged Allocation.



Simple Paged Allocation

Memory space

here page size is 4

Pg-0	0	a
	1	b
	2	c
	3	d
Pg-1	4	e
	5	f
	6	g
	7	h
Pg-2	8	i
	9	j
	10	k
	11	l
Pg-3	12	m
	13	n
	14	o
	15	p

Logical memory

Page no.	Frame no.
0	5
1	6
2	1
3	2

Page table

0	
4	i
	j
	k
	l
8	m
	n
	o
	p
12	
16	
20	a
	b
	c
	d
24	e
	f
	g
	h
28	

Physical memory

Page no. - P

Frame no. - F

offset - d

$$5 \times 4 + 0 = 20 = a$$

$$5 \times 4 + 3 = 23 = d$$

$$6 \times 4 + 0 = 24 = e$$

Paging example for a 32-byte memory with 4 byte pages.

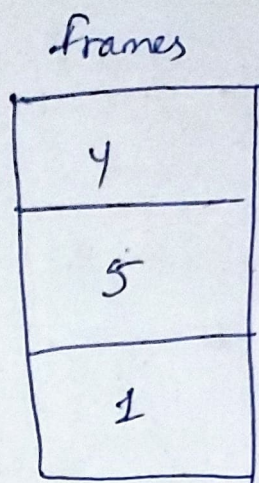
Page Replacement Algorithm :

FIFO

LRU

LFU

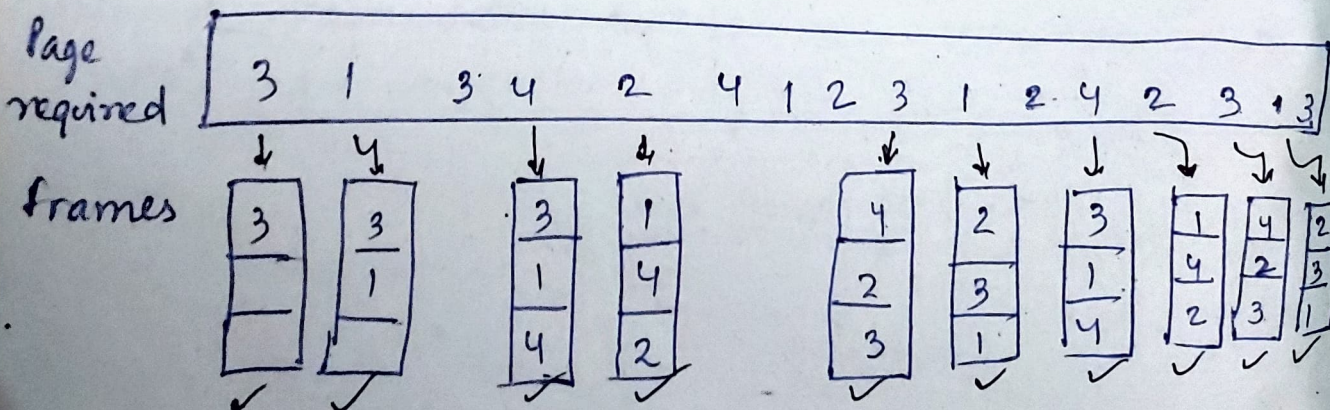
Page 3
is not
present
∴ h/w ~~at~~
support
system
will swap
out any other
page to fill
page - 3



→ this is page replacement algo.

Page Request Sequence - 3 1 3 4 2 4 1 2 3 1 2 4 2 3 1 3

FIFO Assume frame size 3 and no frames preloaded.



10

No. of

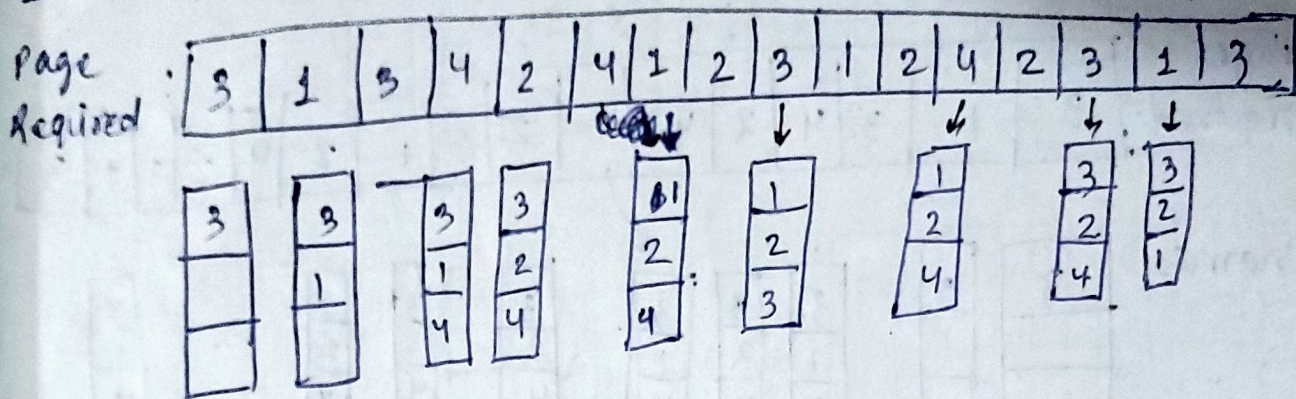
→ Page fault (Page miss) → 10

→ Page replacement → 7 → (10 - 3)

10 - 3

↓
Available frames

B
 Page request sequence: 3 1 3 4 2 4 1 2 3 1 2 4 2 3 1 3
LRU (Least Recently used)



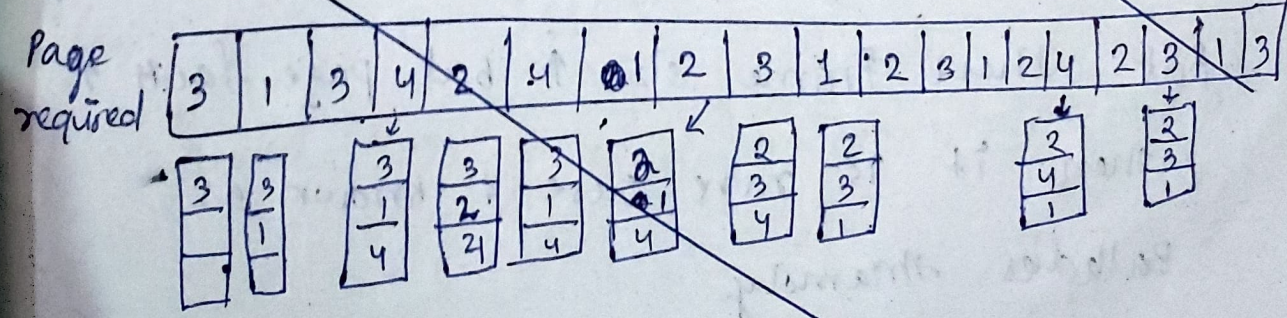
least recently used → Replace the one not used recently.

1 3 4 2
 ↓
 replace

3 4 2 4 1
 ↓
 replace

No. of
 → Page fault → 9.
 → Page replacement → ~~9-3~~ 9-3 = 6

B
 Page request sequence - 3 1 3 4 2 4 1 2 3 1 2 3 1 2 4 2 3 1 3
Lfu (Least frequently used) ~~(FIFO) if tie~~

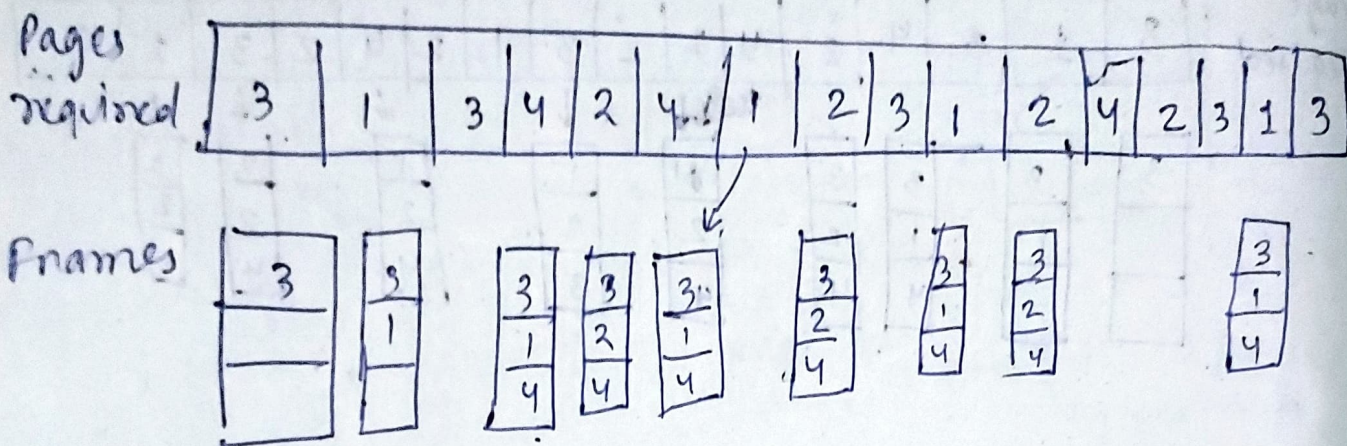


No. of page fault → 10
 No. of page replacement → 7

Q.

Page Request sequence: 3 1 3 4 2 4 1 2 3 1 2 4 2 3 1 3

LFU (Use FIFO in case of tie)



Frequency count for 1, 2, 3, 4

1	0	1	1	1	0	0	1	0	0	1	0	0	0	0	1	1
2	0	0	0	0	1	1	0	1	1	0	1	1	2	2	0	0
3	1	1	2	2	2	2	2	2	3	3	3	3	3	4	4	5
4	0	0	0	1	1	2	2	2	2	2	2	2	3	3	3	3

No. of pages fault = 9

No. of page replacement = 6

If Frame size $\uparrow$ then page fault dec.

But when Frame size $\uparrow$ but page fault $\uparrow$
 then it is rare case & known as
 Belady's Anomaly

Q. Using FIFO

Page required:

1	2	3	4	1	2	5	1	2	3	4	5
---	---	---	---	---	---	---	---	---	---	---	---

Frame 3:

1	1	1	2	3	4	1		2	5
	2	2	2	4	1	2		5	5
		3	4	1	2	5		3	4

Frame 4:

1	1	1	1	2	3	4	5	1	2
	2	2	2	3	4	5	1	2	3
		3	3	4	5	1	2	3	4

In case of Frame 3

→ Page fault = 9.

Page replacement = $6 = (9 - 3)$

In case of Frame 4

→ Page fault = 10

Page replacement = $10 - 4 = 6$

Paging

↓
Internal frag.

Segmentation → Ext. fragmentation

Paging + Segmentation → NO frag.

Segmentation →

• Inter related parts are kept together and

↓
(function, Union, array, pointer) &

made as one segments

& (subroutine
Data area) → where data will be stored

→ Segment is a grouping of information that is treated as a logical entity.

→ A process is divided into different segments each of its own length.

EX: One segment can correspond to a single Subroutine or a group of closely related Subroutines.

- Segmentation uses unequal chunks. (using bound/fence register)
- Each individual segment can be protected.
- Segmentation support the user view of memory.

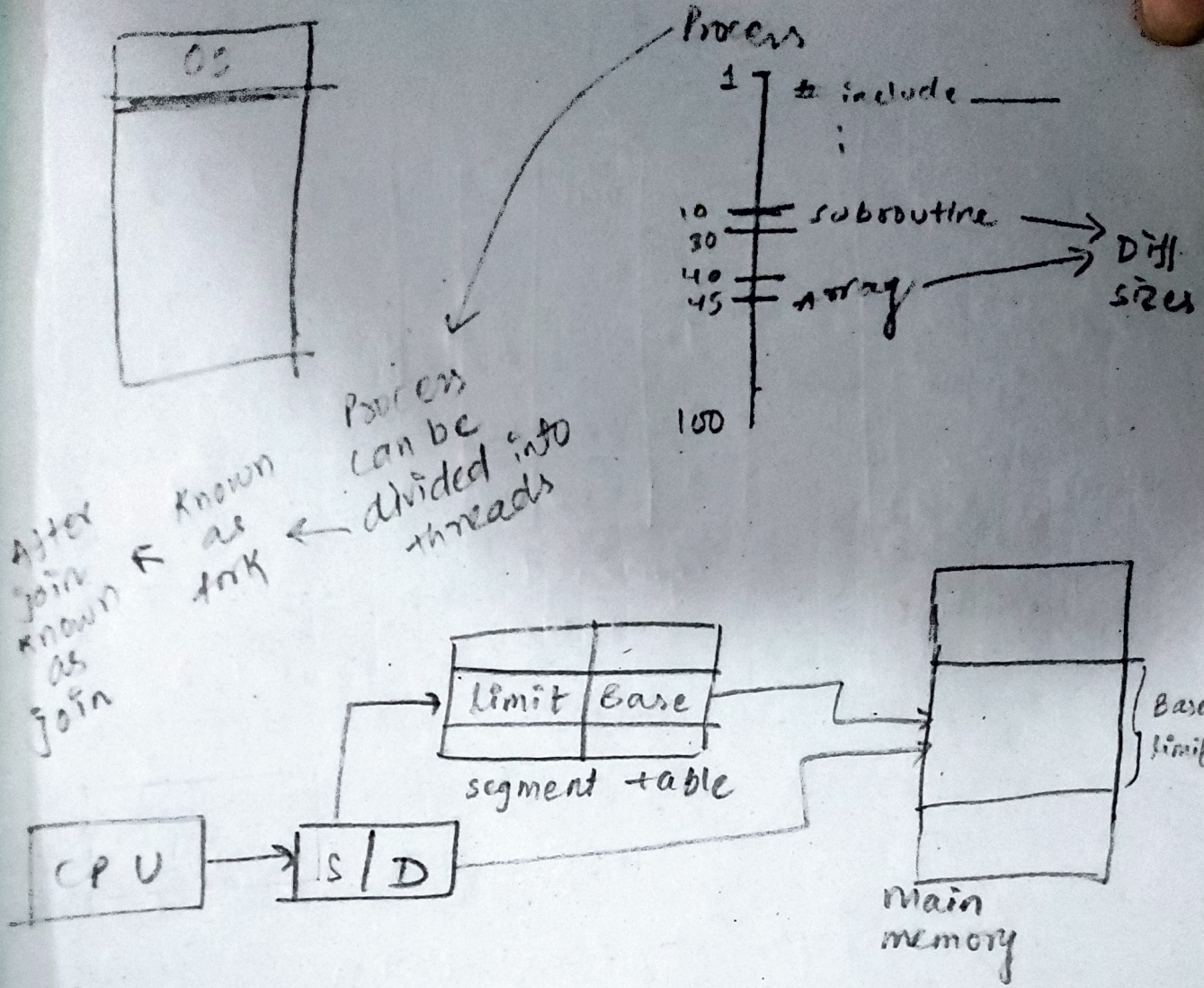
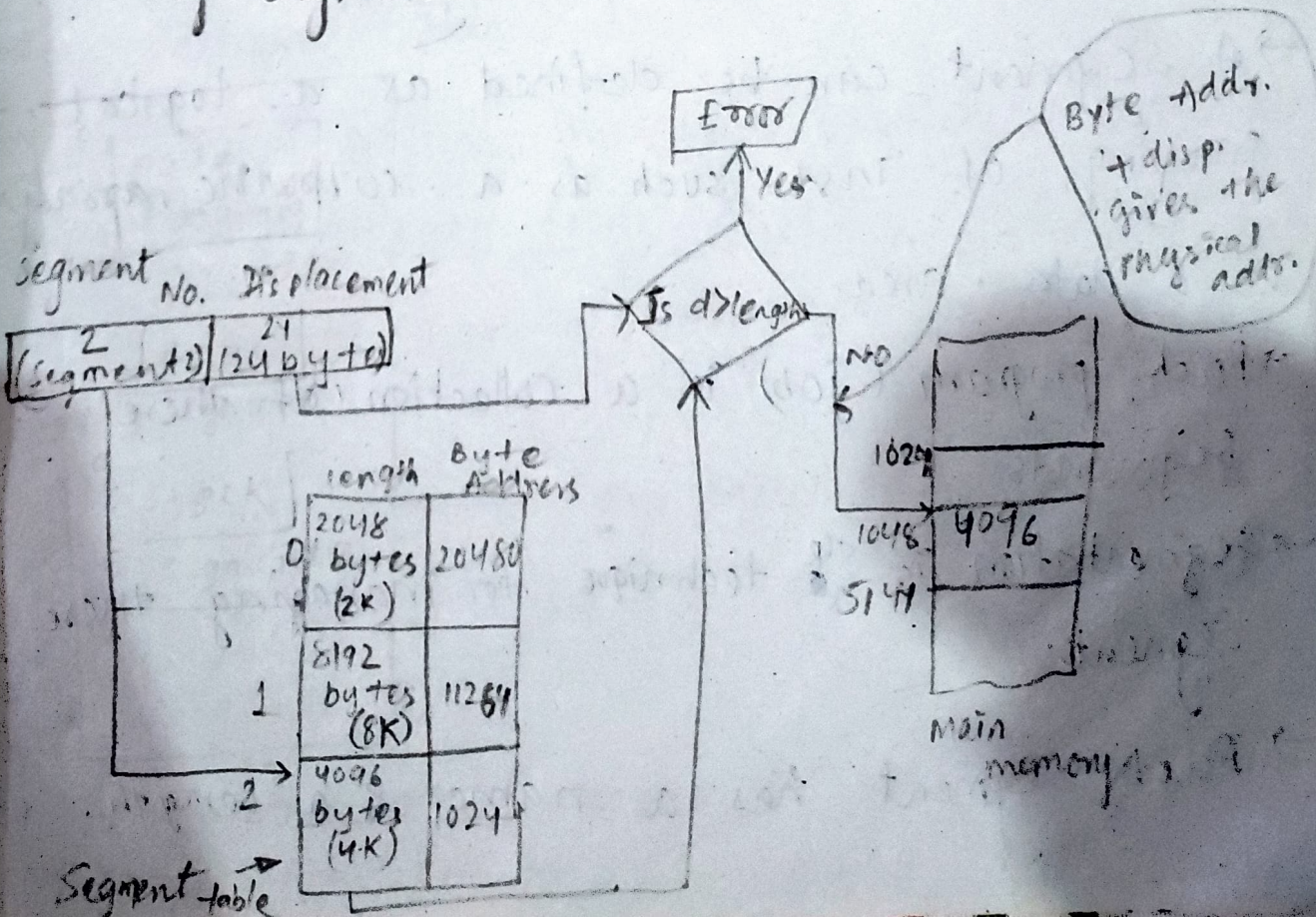
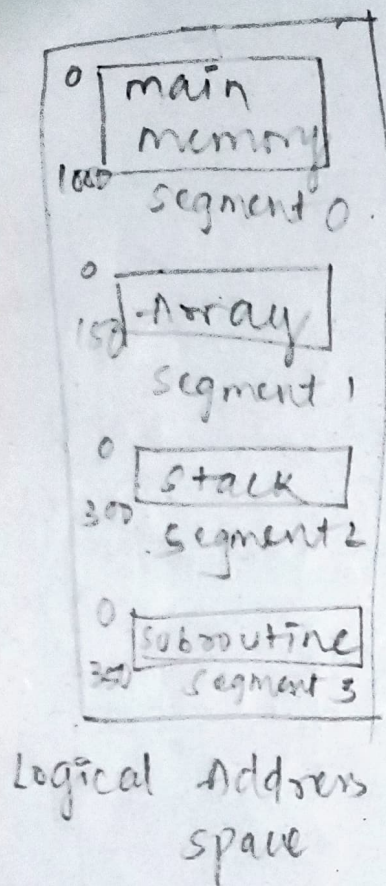


fig Segmented hardware





Seg no.	Limit	Base
0	1000	1500
1	150	900
2	300	2500
3	350	3900

Segment Table

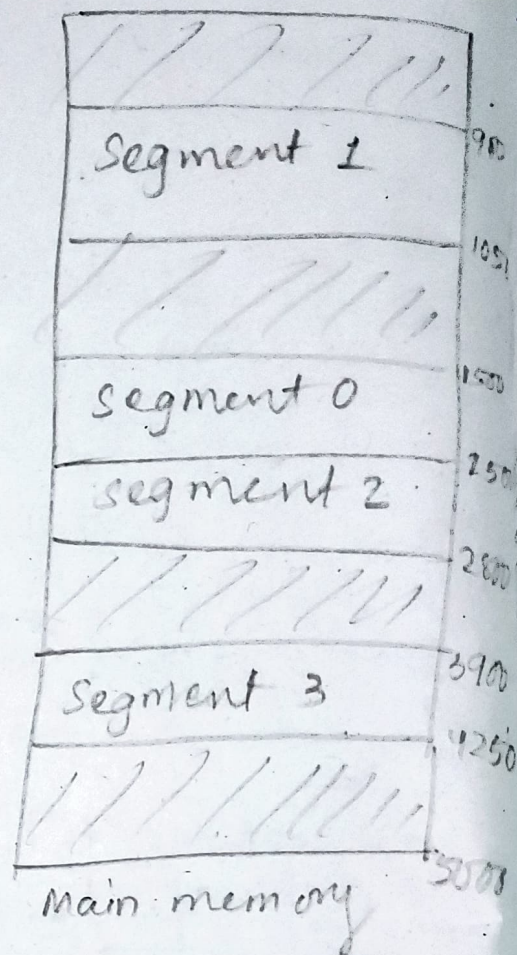


Fig. Example of segmentation

→ A segment can be defined as a logical grouping of inst. such as a subroutine, array or a data area.

→ Each program (job) is a collection of these segments.

→ Segmentation is ^{the} technique for managing these segments.

→ Each segment has a name & a length.

- The address of the segment specifies both segment name and an offset within the segment.
- The segment Base contains the starting physical address where the segment resides in the memory.
- The segment limit specifies the length of the segment.
- The logical address consists of two parts. i.e. a segment no. (s) & the offset into the segment (d) (displacement).
- The segment no. s is used as an index into the segment table.
- The logical address space (job) is divided into 4 segments; numbered from 0 to 3.
- Each segment has an entry into the segment table.
- The limit specifies the size of the segment & the base specifies the starting address of the segment.
- For ex: Segment 0 is loaded in main memory from 1500 KB to 2500 KB. So 1500 KB is the Base

and $2500 - 1500 = 1000\text{KB}$ is the Limit.

Adv.

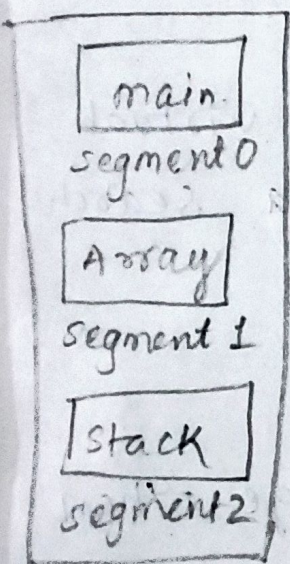
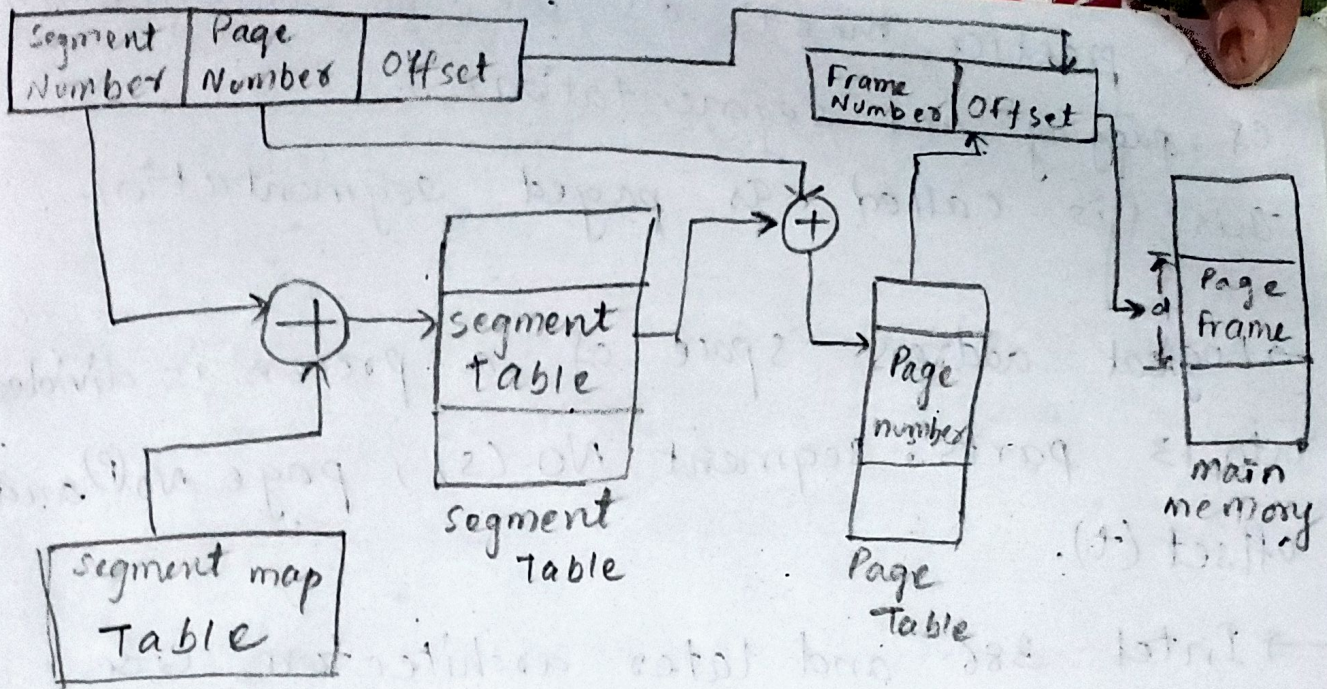
- Eliminate fragmentation
(By moving segments around, fragmented memory can be combined into a single free area)
- Segmentation supports virtual memory concept.
- Allow dynamically growing segments.
- facilitates shared segment.
- Dynamically linking & loading segments:

Disadv.

- Time consuming
- Critical h/w required for garbage collection & compaction

Paged Segmentation :-

- Requires two-level translation; segment tables to page tables and then to main memory.
- Paging leads to internal fragmentation.
- Segmentation leads to external fragmentation.
- trying to reduce external fragmentation increases internal fragmentation & vice versa



Page No.	Frame No.
0	5
1	7
2	9
3	11

Page Table for Seg. 0

Page No.	Frame No.
0	2
1	4
2	6
3	8

Page Table for Seg. 1

Page No.	Frame No.
0	12
1	13
2	14
3	17

Page Table for Seg. 2

(0,0)	0
(0,1)	1
(0,2)	2
(0,3)	3
(1,0)	4
(1,1)	5
(1,2)	6
(1,3)	7
(2,0)	8
(2,1)	9
(2,2)	10
(2,3)	11
(3,0)	12
(3,1)	13
(3,2)	14
(3,3)	15
(4,0)	16
(4,1)	17

Physical address space

↓
Dynamically storing

↓
No. fragmentation

→ In practice most OS use the combination of paging and segmentation.

This is called as paged segmentation.

→ Logical address space of a process is divided into 3 parts: segment No. (s), page No. (p) and offset (t).

→ Intel 386 and later architecture use segmentation with paging.

→ Suppose a process ~~at~~ references a virtual or logical address $V = (s, p, d)$ then a search will be performed to locate (s, p).

There are 3 possibilities for it:

① If (s, p) found in associative storage then we find page frame and then move as per offset to get real memory address.

② If (s, p) is not found in associative storage then we should follow these steps:

Step 1 - Find Base address B of segment ^{mapped} ~~page~~ table (in primary memory)

Step 2 - Add B to the segment no. (say V said s).

Step 3 then we add sl to P to get frame no.

Step 4 Finally we add offset d to get real memory address

(iii) If segment mapped table indicates that segment s is not in primary memory ~~a segment~~ a segment fault is generated.

DISK & DRUM SCHEDULING :-

Q. Suppose the head of a moving hard disk with 200 tracks (0-199) is currently serving a request at track 143 & has just finished a request at track 125. If the queue of request is kept in FIFO order 86, 147, 91, 177, 94, 150, 102, 175, 130. What is the total head movement to satisfy the request for the following disk scheduling algorithm.

(i) FCFS (ii) SSTF (iii) SCAN (iv) C-SCAN (v) LOOK

(vi) C-LOOK

For the above algorithm find out in which algorithm we have starvation problem & find out the policy or technique to overcome this.

SCAN → thoroughly going line by line

LOOK → overlook → not line by line

~~CIRCULAR~~ SCAN → circular scan → starting to end line by line and end to starting again line by line

~~(Asc. order)~~
SSTF → shortest seek time first

FCFS → 86, 147, 91, 177, 94, 150, 102, 175, 130

Ascending order →

86, 91, 94, 102, 130, 143, 147, 150, 175, 177

FCFS

Head movement

143 - 86

57

86 - 147

61

147 - 91

56

91 - 177

86

177 - 94

83

94 - 150

56

150 - 102

48

102 - 175

73

175 - 130

45

Avg: $\frac{565}{9} = 62.77$ cylinders

86, 91, 94, 102, 130, 147, 150, 175, 177

(ii)

SS TF

Head movement

143-147

4

147-150

3

150-130

20

130-102

28

102-94

8

94-91

3

91-86

5

86-175

89

175-177

2

Avg: $162/9 = 18$ cylinders

86

(iii)

SCAN

91

143-130

94

130-102

102

102-94

130

94-91

147

91-86

150

86-0

175

0-147

177

147-150

150-175

175-177

Head movement scan Head movement

13

143-147

4

28

147-150

3

8

150-175

25

3

175-177

2

5

177-199

22

86

199-130

69

147

130-102

28

3

102-94

8

25

94-91

3

2

91-86

5

$\frac{320}{9} = 35.55$

~~320~~ cylinders

$\frac{169}{9}$

$= 18.77$

cylinders

IV	C-SCAN	Head Movement	BC-SCAN	Head Movement
86	143-147	4	143-130	13
91	147-150	3	130-102	28
94	150-175	25	102-94	8
102	175-177	2	94-91	3
130	177-199	22	91-86	5
147	199-0	199	86-0	86
150	0-86	86	0-199	199
175	86-91	5	199-177	22
177	91-94	3	177-175	2
	94-102	8	175-150	25
	102-130	28	150-147	3
		<u>385</u>		<u>384</u>
		$\frac{385}{11} = 35 \text{ cylinders}$		$\frac{384}{11} = 35.818 \text{ cylinders}$

V	LOOK	HEAD MOVEMENT	LOOK	HEAD MOVEMENT
	143-147	4	143-130	13
	147-150	3	130-102	28
	150-175	25	102-94	8
	175-177	2	94-91	3
	177-130	47	91-86	5
	130-102	28	86-147	61
	102-94	8	147-150	3
	94-91	3	150-175	25
	91-86	<u>5</u>	175-177	<u>2</u>
	$\frac{125}{9} = 13.88 \text{ cylinders}$		$\frac{148}{9} = 16.44 \text{ cylinders}$	

As cylinder 75 is less in 1st case & hence is the best movement for rotation data.

(VI)

<u>CLOCK</u>	<u>HEAD MOVEMENT</u>	<u>CLOCK</u>	<u>HEAD MOVEMENT</u>
143-147	4	143-130	13
147-150	3	130-102	28
150-175	25	102-94	8
175-177	2	94-91	3
177-86	91	91-86	5
86-91	5	86-177	91
91-94	3	177-175	2
94-102	8	175-150	25
102-130	<u>28</u>	150-147	<u>3</u>
	$\frac{169}{9} = 18.7$ cylinders		$\frac{178}{9} = 19.77$ cylinder

File Management : ↗

Rows — Tuples, Columns — Attributes

Sl. No.	Reg. No.	S-Name	Branch	Ph. No.	Email	^{sem} release
→ 1						
→ 2						
→ 40						

Primary key → used to identify the rows ^(or records) distinctly.

Candidate key → many keys eligible to become primary key.

Data base → collection of ^{inter} related data.

↳ (collection of data)

It is a collection of stored and operational data used for some specific application or purpose

Stored & operational data → ?

↓
storing the data

which is specific to a particular database.

Database

↓
File

↓
Records

conti. stored data → sequential search

Non-conti. stored data

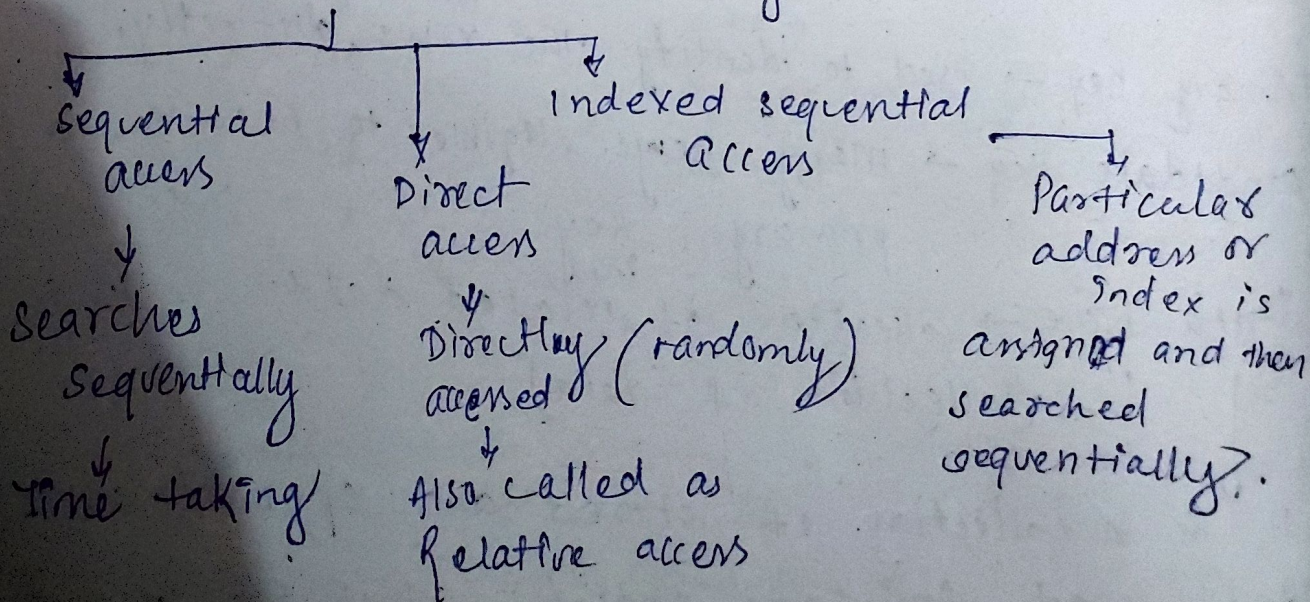
↓
To retrieve data directly

↓
Index search
(Address is given to particular data)

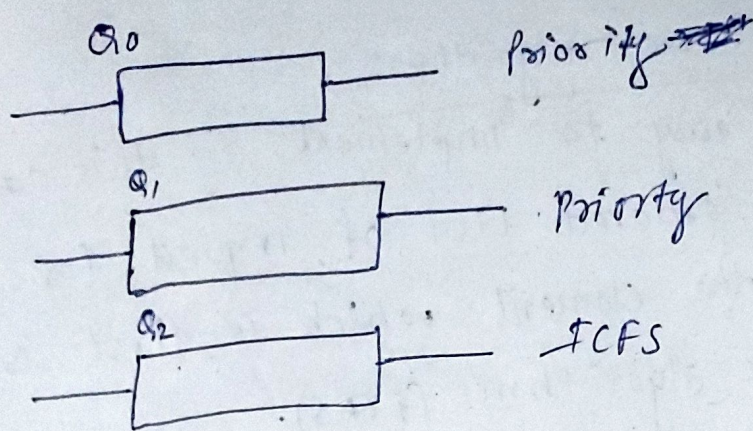
Retrieve all → select * from students

Retrieve one → select * from students where
student_name = "Jigyansha";

Types of files accessing



Multilevel Feedback Queue scheduling:-



Choosing a queue from a set of queue

ECB } Q_1
 EEG } Q_2
 Q_2 } Q_3

Similar to Round Robin but ^{here} ~~the~~ processor will move to the lower priority job which was time taking.

$$\text{priority} \propto \frac{1}{\text{CPU burst time}}$$

DISK 'N' DRUM SCHEDULING:-

(I) First come First Serve algorithm:

It is a simplest and easy to implement, In this case we maintain an ordered list of request for track. In this list the element which is first is served first by the algorithm (FCFS).

(II) Shortest Seek Time First (SSTF):

It serves the request which is minimum seek time from the current held position. Similar to SJFA (Shortest job first Algo). It causes starvation for some request.

(III) SCAN:

In this technique IN/R head starts at one end of the disk and moves to the other end and serve the request when it reaches the required track. From the other end of the direction of movement of the head is just reverse. When a request arrives for a track just in front of the current position of the head it serves by the head immediately. This algorithm is known as elevator algorithm.

(IV) C-SCAN:

It is known as circular SCAN. The head moves from one end to other end of the disk and

serves the request on the way while reaching on the other end it return back to the starting track without serving any request on the return trip.

(v) Sector queue unit:

This scheduling algorithm is used for fixed head device, in this case the track is divided into no. of sectors. Each request for read or write, specify the track and sector. Since seek ~~into~~ time is zero for fixed head device the main service time is the latency time. The algorithm defines separate queue for each sector any request arise for sector I.

As sector I comes under w/r head all the request present in the queue are served.